Abstract

Behavior modeling for military applications needs to consider systems in which all kind of entities participate – machines, humans, human organizations (like platoons or companies) as well as such complex entities like countries, industries and societies. The variety and the structure of entities participating in behaviors in the military domain require the use of representations and tools appropriate for this kind of complexity. Ontological modeling seems to be the best match for this domain. However, there are no known results in the literature on modeling and tracking of behaviors using an ontological approach in which automatic inference over the dynamic models of behaviors can be carried out using inference tools.

A behavior model can be conceptualized in a number of ways - as an abstract concept that is independent of any physical or conceptual entity, as a feature of a specific entity, or as an abstract concept that is associated with one or more physical or conceptual entities. Various knowledge representation mechanisms including State Machines, Hidden Markov Models, Petri Nets, Game Theoretic Models and Bayesian Networks have been used extensively for behavior modeling. Most of the studies have been focusing on modeling behavior of a specific type of entity. For instance, organizational behavior modeling considers an organization as a system of interrelated entities (humans) and then develops models for behavior of humans within an organization.

In the approach presented in this document, behavior is treated as being associated with a *situation*, i.e., with a number of objects (e.g., an organization) being in some relations with each other. While situation objects will normally have some basic behaviors associated by default, they will be able to participate in complex behaviors involving multiple situation objects. Those complex behaviors can occur *in a situation,* and not just as inherent features of a specific object. Thus behaviors are treated as situation objects. In this project we have developed an ontology for situations and then extended it to represent abstract behaviors of situational objects.

The new approach to behavior modeling requires the development of models, techniques and tools that can support both the analyst and the developer in the process of employing this new concept in operational scenarios. With such tools, not only will the analyst be able to employ a system for monitoring whether a specific situation has occurred, but also to *track* situations. While the term ‘tracking situations’ has been used in the information fusion community, it has been used primarily in the sense of generating indicators and warnings when the situation occurs. In the concept presented in this report, behavioral situations are considered as dynamic entities having states, with transitions from one state to another resulting from actions executed by entities participating in a specific behavior that are either inside or outside of the situation being tracked. In this project we have conceptualized various tools, including tools for situational behavior modeling, tracking and learning.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
30 JUN 2009		**20-04-2006 to 31-03-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Situational Behavior Modeling	**FA9550-06-C-0025**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **VIStology, Inc.,5 Mountainview Drive,Framingham,MA,01701**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**95**	

Table of Contents

Personnel Involved in Research Effort

1. Christopher J. Matheus – Principal Investigator
2. Brian Ulicny
3. Mieczyslaw M. Kokar
4. Kenneth Baclawski
5. Won Ng
6. Lena Lau
7. Jerzy Letkowski
8. Jerzy Weyman
9. Robert Dionne
10. Douglas Parent

Publications Stemming from Research Effort

1. Mieczyslaw M. Kokar, Christopher J. Matheus and Kenneth Baclawski. "Ontology-Based Situation Awareness." International Journal of Information Fusion, Vol. 10, pages 83-98, 2009.
2. M. M. Kokar and G-W. Ng. High-level information fusion and situation awareness. Information Fusion, Vol. 10, pages 2-5, 2009.
3. B. Ulicny, C. J. Matheus, M. M. Kokar, and G. M. Powell. Problems and prospects for formally representing and reasoning about enemy courses of action. In Proceedings of the Eleventh International Conference on Information Fusion, Fusion'08. ISIF, 2008.
4. M. M. Kokar, J. J. Letkowski, R. Dionne, and C. J. Matheus. Situation tracking: The concept and a scenario. In Situation Management Workshop: SIMA'08. IEEE, MILCOM, 2008.
5. M. M. Kokar, K. Baclawski, and H. Gao. Category theory based synthesis of a higher-level fusion algorithm: An example. In Proceedings of the Ninth International Conference on Information Fusion, Fusion'06. ISIF, 2006.

Statement of Objectives

The objectives of the proposed research effort were:

1. Analyze the concept of *situational behavior* and provide appropriate formalizations, explanations and prototypical examples that can be understood in the context of situation awareness and higher-level information fusion.

2. Develop a framework for modeling behavioral situations that includes a modeling language that is an extension of a language for modeling situations in general.

3. Conceptualize interactive tools that could support the analyst in developing models of situational behaviors. The tools should support both model building operations and model learning operations. In the latter case, the analyst would provide just feedback or reinforcement to the tool instead of direct instructions for model modification.

4. Conceptualize an approach and tools for incorporating uncertainty into the behavior modeling framework.

5. Propose a mechanism for predicting future behaviors.

6. Provide a proof-of-concept for the situational behavior modeling framework and assess the validity and the utility of the approach.

1 Outline and Significance of the Problem

Situation Awareness has recently become the attention of various research efforts, marking a step forward from the previous focus on Level 1 information processing and fusion, as defined by the JDL Model [1, 2]. However, most approaches so far have treated situations as simply relations, i.e., the meaning of 'situation awareness' is essentially limited to knowing whether a particular relation (relevant to a goal) among some objects holds or not. This interpretation is consistent with Endsley's definition of situation awareness [3]. It is clear, however, that there is a need for a much more refined and innovative view of the term 'situation', i.e., a view in which situations should be considered as entities that can affect, exhibit and participate in various behaviors. This constitutes a new challenge to the information fusion community. The handling of this challenge requires a clear understanding of the meaning of a new concept – 'behavioral situation', referring to the notion of a situation that has a behavior, as opposed to a situation being just a static collection of objects and relations among them. We then can use the term 'situational behavior' to refer to a collective response to an action by a collection of objects in a situation. This in turn leads to the notion of 'behavioral situation modeling' that refers to the activity of modeling situational behaviors.

The new approach to behavior modeling requires the development of models, techniques and tools that can support both the analyst and the developer in the process of employing this new concept in operational scenarios. The new techniques and tools for handling behavioral situations would add new capabilities to the tools of the analyst. Not only would the analyst be able to employ a system for monitoring whether a specific situation has occurred, but also the situation would be tracked. While the term 'tracking situations' has been used in the information fusion community, it has been used primarily in the sense of generating indicators and warnings when the situation occurs. In the concept presented in this report, behavioral situations is considered as dynamic entities having states, with transitions from one state to another resulting from actions executed by entities participating in a specific behavior. Situation tracking would then have a clear sense, similar to tracking of dynamic objects in Level 1 fusion.

To explain the idea of situation tracking, consider an example of a scenario in which the tracked situation is "Iraq is threatening Kuwait". The top-level entities here are two countries – Iraq and Kuwait. The situation is labeled as 'threatening'. While knowing whether it is the case that Iraq is threatening Kuwait is important for the process of making decisions on how to respond to the situation, it is much more important to understand more details about this situation. The concern here is that the situation type of 'invading' may follow the situation type of 'threatening'. Thus a situation type is like a state that can be changed by switching to another state. The situation can be considered as having a number of internal states (sub-states) and the transition to the new situation type is more likely in some sub-states of 'threatening' than in others. For instance, at some time Iraq was in the sub-state of 'threatening' that can be labeled as 'diplomatic accusations'. The actions that led to this sub-state were things such as: 'Iraq demands that Kuwait forgive debts owed by Iraq after the Iraq-Iran war' and 'Iraq accuses Kuwait of overproduction of oil, claiming that Iraq's oil has been "stolen" by Kuwait'. Another sub-

state that could be a component of a model of the threatening behavior could be labeled as ‘forces deployed along the boarder’. The actions that lead to this state are movements of particular military units from positions in other parts of the territory to the Kuwait boarder. Note that although each of these sub-states is associated with the ‘threatening’ situation, the severity of threatening, the likelihood of switching to the state of ‘invading’ and the selection of methods of responding to the situation are different. It is clear that tracking situations is important to both the selection of appropriate courses of action and to the success of a specific operation.

2 Research Objectives

The objectives of this research effort were:

1. Analyze the concept of *situational behavior* and provide appropriate formalizations, explanations and prototypical examples that can be understood in the context of situation awareness and higher-level information fusion.
2. Develop a framework for modeling behavioral situations that includes a modeling language that is an extension of a language for modeling situations in general.
3. Propose interactive tools that can support the analyst in developing models of situational behaviors. The tools should support both model building operations and model learning operations. In the latter case, the analyst would provide just feedback or reinforcement to the tool instead of direct instructions for model modification.
4. Conceptualize an approach and tools for incorporating uncertainty into the behavior modeling framework.
5. Propose a mechanism for predicting future behaviors.
6. Provide a proof-of-concept for the situational behavior modeling framework and assess the validity and the utility of the approach.

3 Work Plan

In this section we describe a top-level view of our approach to situational behavior modeling. We first discuss the notion of ‘behavior’, as it is presented in various publications. Then we introduce a number of concepts that are used as the main constructs of a situational behavior modeling framework. This is followed by the discussion of tools that are proposed to be developed to support the framework. We believe that behavior needs to be modeled in a context, i.e., first an ontology of entities that can participate in a behavior needs to be developed, and then behaviors are added to such an ontology as an extension of the situation awareness modeling mechanism. Consequently, the tools for modeling behaviors have to be compatible with those used in ontological modeling. However, special features specific to behavior modeling are needed. In particular, situation monitoring tools should be extended to include the capability of behavior monitoring. Since behavior modeling and behavior monitoring tools involve a relatively high level of complexity, mechanisms will be needed to help (or guide) the user so that a model can be constructed in an interactive manner in which the tools play a proactive role of suggesting various options to the user. The next step of complexity is adding the learning capability to such tools. And finally, since both the events that will affect the monitoring of behaviors and the rules of transition from one

situational state to another are non-deterministic, the modeling and tracking framework needs to provide means for specifying uncertainty levels associated with particular events, actions and decisions and for computing uncertainty associated with particular system decisions. We will present our proposal for the handling of uncertainty in this section. Behavior prediction will be achieved by posting queries to the model and invoking the query answering mechanism.

The rest of the report is structured in the following way. First, in Section 4 we provide some introductory discussion about the notions of "behavior" and "behavior modeling". Then in Section 5 we describe an ontology of situations and behaviors. This is followed by a formalization of the problem in mathematical terms and a presentation of the Especs approach that was chosen as a way of solving our problem. Section 6 gives an overview of various theoretical approaches and a literature survey of other suggested solutions addressing the challenges posed by this project. Section 8 presents the application of Especs to the GCD problem to illustrate its use for situation modeling. Section 9 shows how the proposed approach is used in a concrete situation tracking scenario and describes the practical implementation of the solution using various tools, like ConsVISor, BaseVISor, Specware and Python. Section 10 describes a conceptualization of tools to support the analyst in the task of situation tracking. And finally, Section 11 presents conclusions and directions for the continuation of this approach.

4 Behavioral Situation and Situational Behavior Modeling

The main subject of this project is an approach to behavioral modeling. Since the notion of "behavior" is at the center of this work we first need to clarify what we mean by this term and put the concept in context. The term 'behavior' has various definitions. Below we provide two of them.

> **behavior, behaviour** -- (psychology) the aggregate of the responses or reactions or movements made by an organism in any situation [4].
> **behavior** – the way a person behaves or acts; conduct; manners; an organism's (…) responses to stimulation esp. those that can be observed; the way a machine, element, etc. acts or functions [5].

As we can see, the term 'behavior' is defined in the context of an *object* (organism, person, machine, element) and some *activity* (the first definition calls it 'situation', the second one refers to 'acting' and 'functioning') and *actions* (or functions).

In Figure 1 we present (in UML notation [53]) the meaning of behavior in context. In this figure, behavior is captured by a model, called here Behavior Model. A model (i.e., an instance of the class Behavioral Model) defines an activity. This is represented by the property *definedBy* between an activity and a behavioral model. Figure 1 also states that a behavioral model sequences actions (the *sequencedBy* property), where actions are *part-of* activities (indicated by the diamond at the tail end of the association between Activity and Action). Activities have object-participants (the *participant* property of Activity). The annotation 1..* at the arrow head of the *participant* property means that for each activity there must be at least one object-participant. At the same time, a given object can

be involved in any number of activities, but possibly none (indicated by the 0..* multiplicity of the *involvedIn* property). Each object may have some capabilities of executing (*canPerform*) actions. Objects can have a number of such capabilities, but it is also possible that some objects are passive, i.e., their set of capabilities is empty.

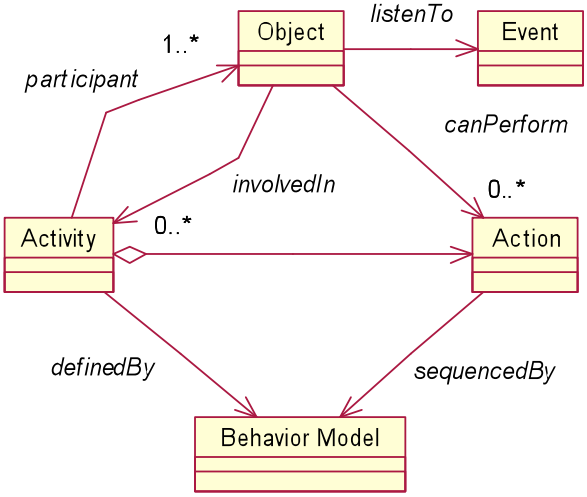


Figure 1. Behavior Context

An example of such a model can be a State Transition Diagram (STD [53]). In Figure 2 the STD class is shown as a subclass of Behavior Model. STDs are not the only possible kind of modeling formalism that can be used to capture behaviors. Other examples include sequence diagrams [6], influence diagrams [7], collaboration diagrams [6], Bayesian Networks [8]. Figure 2 refines the modeling formalism of Figure 1 by adding three additional modeling elements: STD, State and Transition. Since STD is a subclass of Behavioral Model, it inherits the modeling elements from Figure 1. Using STDs, behaviors are modeled as sequences of transitions between states. Each transition is triggered by an action, which in turn is the result of listening to events by the object that executes the action. The two diagrams do not show all the details of the modeling formalism. The formalism will be refined later on in this document.

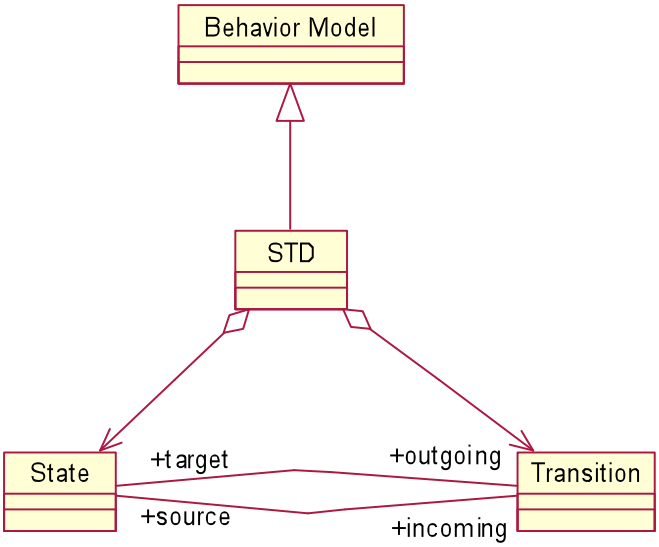


Figure 2. State Transition Diagrams

5 An Ontology of Situations and Behaviors

5.1 An Introduction to Situations

Situation awareness was envisioned as the main part of Level 2 processing in the JDL model [1,2] But only recently has it become the center of attention for information fusion research. As is typical with a new field of research, various studies on this subject have contributed results that are difficult to integrate into one coherent conceptual structure. In other words, the field of situation awareness needs a unifying framework that would play the role of a common theory integrating various research efforts.

Situation awareness research can be classified by the subject that performs this process - human or computer. For human situation awareness, the model proposed by Endsley [3] has been more or less accepted by the information fusion community. Moreover, this model has been used in various studies as a justification for structuring the computer-supported situation awareness process. While the human situation awareness model has been grounded in various studies of cognitive science, the computer situation awareness process still lacks a more systematic treatment. Moreover, the difference between human and computer processing is that the human situation awareness process needs to be measured and possibly supported, which is the main focus of Endsley's research, while the computer process needs to be defined and implemented.

Clearly it is necessary to develop unambiguous specifications, designs and implementations of situation awareness processes. One of the trends in this direction that became prevalent in recent years is that of using ontology-based computing as a paradigm on which to develop computer based situation awareness processes. Although all of these efforts are based on ontologies as the main representational structure, they lack commonality in the repertoire of concepts used in the analysis and the synthesis of situation awareness processing.

Artificial Intelligence (AI) has dealt with a notion of ``context'', which, according to Akman and Surav [9], stands for the same concept as ``situation''. This line of AI research was started by McCarthy (cf. [10]) and is still an active research field. The main idea of the AI approach is to introduce a predicate, *isp(c,p)*, that explicitly states the fact that the proposition *p* is true in the context *c*.

Sowa in his book [11] provides both a historical overview of the AI treatment of context and an approach to representing contexts (situations) in the formalism of *conceptual graphs* [12]. Conceptual graphs are patterned upon *existential graphs* developed by Charles S. Peirce. Similarly to McCarthy's approach, Sowa introduces a *description predicate*, *dscr(x,p)*, which captures the fact that the entity *x* is described by the proposition *p*. When the entity is a situation, then the proposition *p* describes that situation. This predicate is then used to state facts that hold in a given situation. Conceptual graphs are representable in a graphical form that is more human friendly than a computer-readable form called Conceptual Graph Interchange Form (CGIF).

The principal goal of this section is to formalize the main concepts of situation awareness using a language that is both processable by computer and commonly supported. To achieve this goal we first need to identify appropriate concepts that can be classified as part of the situation awareness domain. We have already mentioned a number of such concepts provided by Endsley. Another source of information on situation awareness is the *Situation Theory* developed by Barwise and Perry [13,14,15], which was subsequently extended by Devlin [16]. Since the concepts of situation theory encompass most of the concepts discussed by Endsley, and since situation theory is described in a more formal language, here we first provide a short overview of situation theory and then show how situation theory can be captured in a formal language with a computer-processable semantics.

Computer support for logic is a popular theme in computer science, and there are many languages that have been developed for this purpose. Moreover, situation theory has already been expressed in terms of some existing logical languages. However, few of these languages have even been standardized, and fewer still are commonly supported by popular software tools and systems. Currently the only languages that have such support are the languages of the Semantic Web [17]: the Resource Description Framework (RDF) [18] and the Web Ontology Language (OWL) [19], which is based on RDF. OWL improves on RDF by adding many new logical capabilities. One of the most important new capabilities is the ability to define classes in terms of other classes using a variety of class constructors such as unions, intersections and property values. Accordingly, we have chosen OWL as the language for formalizing situation theory, and in this paper we give examples to show how the reasoning techniques pioneered by Barwise and Devlin can be mapped to OWL class constructors.

As mentioned above, situation theory has already been expressed in terms of some existing languages. While the argument in favor of OWL over these other languages is reasonably compelling, it is still worthwhile to consider some of the potential

disadvantages of OWL relative to the alternatives. Two of the most commonly mentioned disadvantages of OWL are that it is wordy and unreadable. In fact, the wordiness of OWL is only a disadvantage from the point of view of people, not computers. To computers it becomes a significant advantage. There are various languages for representing OWL, but all of them share common features such as self-description, decoupling of facts from the containing document, and reduction to simple elementary statements. The first feature allows OWL to be parsed by commonly available generic parsers such as the ubiquitous XML parsers. The latter two features make it much easier to store and manage OWL facts in databases. These advantages easily outweigh the disadvantage of the wordiness of OWL. Concerning the unreadability of the XML representation of OWL, this is also only an issue for people, not computers. It is expected that people would usually neither read nor write OWL using the XML representation. However, it is still necessary sometimes, so it could be argued that some other language would be better. To deal with this problem, a number of alternative OWL syntaxes as well as GUIs have been developed that are much more readable and succinct and that map directly to the XML representation. The Abstract Syntax and N3 are two well known examples of syntaxes, and Protégé is a well known example of a popular GUI and IDE that supports OWL. Furthermore, these notations and GUIs are about as readable as possible given the requirement that the notation be self-describing.

The OWL language has three levels that have progressively richer semantics but are also progressively harder to process. Since situation theory requires that one model ``classes as instances'', it is necessary to use the highest OWL level, OWL Full. Furthermore, while OWL Full is sufficient for nearly all concepts required by situation theory, there are a few that even OWL Full cannot express. Those concepts can be formalized using a computer-processable rule language compatible with OWL such as RuleML. The concepts expressed in OWL and the ones expressed using rules together form a formal *ontology* for situation awareness. Since the intent of our ontology is to capture most of situation theory, we call it the *Situation Theory Ontology*, or STO, for short.

Such an ontology can play the role of a unifying theory of computer-based situation awareness. In this paper we describe all the concepts in this ontology. One of our claims is that STO is compatible with current thinking about situation awareness in the community. In particular, there are clear relations between the concepts in this ontology and Endsley's model of human situation awareness.

While the ontology discussed here has useful characteristics, it is not complete, and experts in this field might have somewhat different opinions on which concepts should be included and how they should be represented. An ontology is valuable only if the majority of the community accepts its main concepts and structure. The most important aspect of our proposal is that the ontology is formally defined, i.e., it is expressed in a language with formal semantics. This fact makes it possible to ground the discussion of the ontology in a precise and unambiguous language.

The secondary goal of our approach is to indicate how the STO can be used to develop situation awareness systems. The major point of this part is that a significant amount of flexibility can be achieved through the use of generic ontology-based tools. To achieve the secondary goal, we give examples of how the ontology based approach to situation awareness can be used. For this purpose we first describe a simple example (somewhat similar to the one used in Sowa's book and show how that situation can be represented. Then we show how automatic logical inference can be carried out using the formal description of the situation and of the ontology.

5.2 Situations and Situation Awareness

A behavior model can be conceptualized in a number of ways - as an abstract concept that is independent of any physical or conceptual entity, as a feature of a specific entity, or as an abstract concept that is associated with one or more physical or conceptual entities. Various knowledge representation mechanisms including State Machines, Hidden Markov Models, Petri Nets, Game Theoretic Models and Bayesian Networks have been used extensively for behavior modeling. Most of the studies have been focusing on modeling behavior of a specific type of entity. Organizational behavior modeling, on the other hand, takes as its starting point the view that an organization is a system of interrelated entities (humans) and then develops models for behavior of humans within an organization [20]. However, as is the case in other approaches, the focus is on one type of entity – the human.

Behavior modeling for military applications needs to consider systems in which all kind of entities participate – machines, humans, human organizations (platoons, companies, battalions, brigades, armies) as well as such complex entities like countries, industries and societies. This makes the task of behavior modeling and analysis extremely complex. The variety and the structure of entities participating in behaviors in the military domain require the use of representations and tools appropriate for this kind of complexity. Ontological modeling seems to be the best match for this domain.

However, we are not aware of any work that would attack the problem of modeling and tracking of behaviors of situations using an ontological approach. We are aware of some papers that deal with the modeling of behaviors, but not with situational behaviors. For instance, one of the papers states that behavioral modeling should use an ontology-based approach [21]; but it was just listed as a topic for future research and no results have been shown.

In our approach, as described in Section 4, behavior is treated as being associated with a number of objects (e.g., an organization). In that case, a behavioral model is an extension to the models of the entities participating in the behavior. Behavioral knowledge is thus added to the knowledge about the entities and their environment. In particular, when domain knowledge is modeled using the ontological approach (i.e., with concepts modeled as classes and properties, and instances modeled as instances of the classes and the properties) then the behavior model must be compatible with the domain knowledge.

When domain knowledge already exists and is captured in an ontological language, like OWL [22], modeling of behavior can be leveraged by such domain knowledge. In other words, a behavioral model is built on top of such knowledge instead of being built from scratch. Additionally, tools can take advantage of this knowledge in their search for guidance that they can provide to the user.

In this project we have developed an ontology of situations [23] and an ontology of abstract behaviors that were intended as extensions of the Core Ontology for Situation Awareness [24]. While situation objects will normally have some basic behaviors associated by default, they will be able to participate in complex behaviors involving multiple situation objects. Those complex behaviors can occur *in a situation*, and not just as inherent features of a specific object. For instance, an aircraft can be "threatening", i.e., its behavior is threatening, but only in a particular situation. An aircraft can be "threatening" if it gets too close to another aircraft, but when it is far away it is not threatening at all. This leads us to believe that behaviors should be (at least partially) separated from any specific physical object, but rather should be treated as situation objects that can then be associated with situations. In other words, we need an ontology of abstract behaviors.

5.3 Situation Theory Ontology (STO)

As a result of this project we have developed an ontology which we call the Situation Theory Ontology, or STO for short. This ontology has been described in [23]. Here we provide only an overview of this ontology for the sake of self-containment of this report.

5.3.1 Situation and Situation Awareness

Although the notion of "situation awareness" is part of the data fusion lexicon (cf. [25]), this term has been used with a number of different meanings. In this section we identify and discuss some of the most common interpretations of this concept and relate them both to the JDL Data Fusion Model [2] and to the model of Endsley [3]. We use Figure 3 to support this discussion.

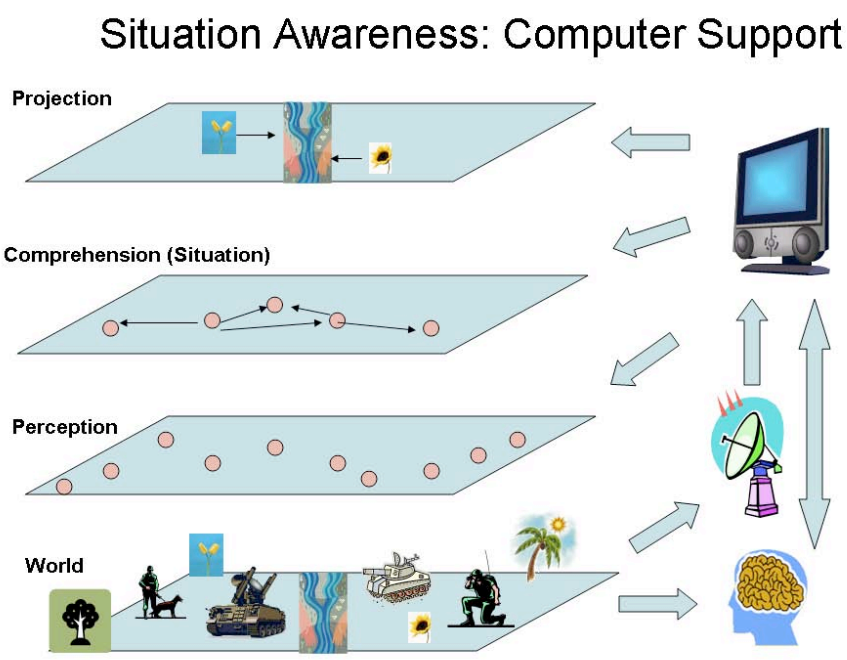


Figure 3. Situations and Perception

Figure 3 shows four planes, each referring to a different level of abstraction. The bottom layer shows the World, i.e., the physical (or abstract) world that is the subject of some inquiry. Although this figure suggests that the World is associated with a geographical region, it actually does not have to be so. It's just a symbolic depiction of the things that give rise to a situation. These can be either physical or conceptual things, or both.
To the right of the World plane, a human head depicts the fact that situation awareness actually takes place in the human's brain. The human observes some aspects of the World, and the human gets inputs from the computer, as shown in the figure.

The next layer is denoted as "Perception." The dots on this plane represent objects from the World that are observed through sensors and represented in computer memory.
The arrow from the World plane toward the radar icon represents the sensory process, which then feeds the computer, which in turn generates the object representations. The label "Perception" represents the fact that this kind of representation is compatible with the output of the Perception process in Endsley's model [3]. In some discussions of situation awareness this kind of representation is considered to be the "situation," i.e., some people consider the situation to be the knowledge of all the objects in a specific area, and possibly their kinematic states.

This is not how this term is defined in dictionaries. For instance, Webster Dictionary [26] defines ``situation'' as:

> 1 a: the way in which something is placed in relation to its surroundings.

Thus the emphasis in the dictionary definition is on relationships. The relations are viewed from the point of view of a thing, and they capture how other things in the surroundings of that thing are related to it; the thing is the focal object of the situation.

The JDL model also recognizes the role of relations as the basic feature of situations:

> Situation Assessment: estimation and prediction of relations among entities, to include force structure and cross force relations, communications and perceptual influences, physical context, etc.

In this paper we consider this kind of situation. In Figure 3 this kind of notion of situation is represented by the plane labeled as ``Comprehension." The lines that connect some of the points represent the relations. Again, this is just a view that symbolizes relations. Although the figure shows only lines connecting pairs of points, i.e., only binary relations, in fact relations can relate more than two objects. Moreover, the same set of objects can be related by many different relations. The label ``Comprehension" indicates that this representation maps to the Comprehension part in the Endsley's model of situation awareness.

Note, however, that although the JDL model captures the role that relations play in the definition of ``situation," it misses the essence of ``awareness" in its formulation. For instance, the term ``aware" provided in Webster's Dictionary is explained as:

> *awareness* implies vigilance in observing or alertness in drawing inferences from what one experiences.

In other words, a subject is aware if the subject is capable not only of observing some objects (experiences) but also of drawing conclusions (inferences) from these observations. The need for inference comes from the fact that not all information comes explicitly through experience. This is particularly true for relations. While it is typical that information about objects (or at least their properties) can be experienced, or observed directly, the relational information must be inferred. This aspect of awareness seems to be part (although not explicitly) of ``comprehension" as defined by Endsley.

The top layer of Figure 3 shows the plane labeled ``Projection." This layer has a direct relationship with Endsley's model in which projection is defined as the capability of anticipating future events and their implications.

The importance of relations and inference for situation awareness can be easily observed in various scenarios in which humans can be said to be aware (or not). For instance, consider a scenario of watching a game, like American football or baseball, by someone who has never learned the rules and the strategies of these games. Although the person can clearly see where each player is and where the ball is, the person still has no idea of ``what is going on" and thus cannot claim to be ``aware" of the situation of the game being watched. The main part of being aware is to be able to answer the question of ``what's going on?" As this example shows, in order to be able to do so, one needs to

have data pertinent to the objects of interest, some background knowledge that allows one to interpret the collected object data and finally a capability for drawing inferences.

The essence of experienced vs. inferred information can also be expressed in formal terms. In mathematics, a relation is a subset of the Cartesian product of a number of sets. For instance, the Cartesian product of two sets A and B is the set of all ordered pairs <a,b>, i.e., $A \times B = \{<a,b> \mid a \in A, b \in B\}$. A relation R is then a subset of the Cartesian product, $R \subseteq A \times B$. A relation can be given (specified) either extensionally or intensionally. An extensional specification of a relation is given by explicitly listing all the tuples of the relation. An intensional specification of a relation, R, is given through a predicate, P. In that case R contains all those tuples r for which the predicate P is true. It is formally written as: $R = \{<a,b> \mid P(a,b)\}$.

Now the question is how we know for a given pair <a,b> that the predicate P is true. This is where the power of inference comes to bear. Predicates are the main component of *sentences*, which in turn are part of *logical theories*. Relations, on the other hand, are part of *models*, i.e., interpretations of sentences. The process of inferencing, or reasoning, is carried out within a specific theory. A computer-based reasoning process is purely syntactic, i.e., an inference engine manipulates ``facts" that are stored as strings. It matches ``inference rules" to patterns in its current fact base and ``derives" new facts according to the inference rule whose pattern has been matched. To make this possible, one must have:

1. A formal language in which all the facts used in the reasoning process are expressed, and
2. A formal specification of the reasoning process.

A formal language is given by a grammar and a notion of interpretation. A *grammar* is given by a number of rules for constructing compound sentences out of elementary sentences. An *interpretation* is a function that maps all of the elements of the formal language to a relational structure, called a *model*. In particular, an interpretation maps each predicate to a relation. A sentence is said to be true for an interpretation if its corresponding relation holds in the model. For a set of sentences A, a model of A is any such relational structure for which every sentence in A holds.

The specification of the reasoning process is given by the notion of *entailment*. A set of sentences A *entails* a sentence s if and only if for every interpretation of A, whenever all sentences of A are true, the sentence s is also true. In the context of situation awareness we will use the term ``entailment" to indicate the process in which a sentence is determined to be entailed by some set of sentences.

In addition to reasoning about relations, situation awareness involves the use of the concept of situation in real life. While a situation can be defined as a set of relations with other objects, both the objects and the relations change with both time and location. For instance, one is in different situations when one is driving home and when one is hiking in the mountains. To make use of situation awareness, especially for decision making, one must be able to recognize situations, assess their impact on one's goals, memorize

situations, associate various properties with particular situations, and communicate descriptions of situations to others. This leads to two additional requirements with respect to the representations of situations:

1. Situations can be classified by Situation Types, and
2. Situations can be treated as objects, like physical objects or conceptual objects.

These requirements support the idea of modeling situations as typed objects within the object-oriented paradigm.

A number of philosophers and logicians introduced concepts similar to that of a situation, including von Mises [27] in 1949 and Bunge [28] in the 1970s. However, the earliest formal notion of situation (although not situation awareness) was introduced by Barwise and Perry as a means of giving a more realistic formal semantics for speech acts than what was then available [13,14,15]. In contrast with a ``world" which determines the value of every proposition, a situation corresponds to the limited parts of reality that we perceive, reason about, and live in. As Barwise explains [15]:

> One of the starting points for situation semantics was the promotionof real situations from second class citizens to first class citizens. By a situation, then, we mean a part of reality that can be comprehended as a whole in its own right - one that interacts with other things. By interacting with other things we mean that they have properties or relate to other things.

While Barwise's situation semantics is only one of the many alternative semantic frameworks currently available, its basic themes have been incorporated into most of the others.

5.3.2 Situation Semantics

We now present a formalization of Barwise's situation semantics in terms of an ontology, with some parts using mathematics and rules. We call the resulting ontology the Situation Theory Ontology (STO). Most of our interpretation of the meaning of situation semantics is based upon Devlin's work [16]. Devlin formalizes a number of concepts developed by Barwise and Perry, subsequently extended by Devlin; we will refer to these concepts as *situation theory*, or the *situation-theoretic framework.*

While we have attempted to be fully faithful to situation theory, it is not possible to give a completely rigorous formalization. As Devlin explained, ``Although described as a `theory,' situation theory is more profitably approached as a set of mathematically-based tools..." Accordingly, our mapping can only be a correspondence, not a formal equivalence.

Barwise and Perry began with the assumption that ``people use language in limited parts of the world to talk about (i.e., exchange information about) other limited parts of the world. They call those limited parts of the world *situations*. Events and episodes are situations in time, scenes are visually perceived situations, changes are sequences of situations, and facts are situations enriched (or polluted) by language." Devlin stresses

that ``the appearance of the word *parts* in the above quotation is significant. Situations are parts of the world and the information an agent has about a given situation at any moment will be just a part of all the information that is theoretically available. The emphasis on partiality contrasts situation semantics from what was regarded by many as its principal competitor as a semantic theory, possible worlds semantics.

5.3.3 Basic Notions and Relationships

In situation theory, information about a situation is expressed in terms of *infons*. Infons are written as: <<R, a_1, …, a_n, 0/1>>, where R is an n-place relation and a_1, …, a_n are *objects* appropriate for R. Since situation theory is multi-sorted, the word ``appropriate" means that the objects are of the types appropriate for a given relation. The last item in an infon is the *polarity* of the infon. Its value is either 1 (if the objects stand in the relation R) or 0 (if the objects don't stand in the relation R). Devlin states that ``infons are not things that in themselves are true or false. Rather a particular item of information may be true or false about a situation." Infons may be recursively combined to form *compound infons* by using conjunction, disjunction and situation-bounded quantification.
Devlin does not have a term for infons that are not compound. We say that such infons are *elementary*.

To capture the semantics of situations, situation theory provides a relation between situations and infons. This relationship is called the *supports* relationship which relates a situation with the infons that ``are made factual" by the situation. Given an infon $\sigma$ and situation s the *proposition* ``s supports σ" is written as: $s \models \sigma$.

The relation between a situation (in the world) and a representation of the situation (in a formal framework) is relative to a specific agent. In situation theory, it is the agent who establishes such a link. This link is defined by *connections* that link entities in the world to formal constructs of the situation-theoretic framework. These connections are not part of the formal theory. Thus they cannot be part of any formal theory such as a situation awareness ontology.
One refers to situations within a formal theory by using *abstract situations*, although the qualifier ``abstract" is usually dropped in most discussions of situation theory, and we will generally do so as well in the following discussion.

5.3.4 The Meaning of Supports vs. Derives

Before we proceed further with our formalization of situation theory, we need to provide some clarification regarding our formalization vs. Devlin's. Since our formalization makes use of OWL, whose semantics is specified in the classical model-theoretic way, our formalization can be claimed to *resemble* situation theory rather than faithfully implement it. To clarify the relationship between the two forms of semantics, we need to discuss the relation between the notion of ``supports" in situation theory and ``models" in model theory. A possible confusion about these two terms comes from the fact that both approaches use the same symbol, $\models$, to describe two different concepts.

In the classical model-theoretic semantics the symbol, |=, stands for ``satisfies." In other words, the meaning of this symbol is that a given relational structure, called the *model*, satisfies a set of sentences. A set of sentences A is said to *entail* a set of sentences B if any model of A contains the model of B. A relation parallel to the entailment relation is called *derives*, or *logically implies*. While entailment is defined on both sentences and models, logical implication is defined only on sentences (i.e., within *theories*). Derivation is defined in terms of *inference rules*. A set of sentences A *derives* a set of sentences B, written as A |- B, if each sentence in B can result in ``true" by the application of the inference rules and other sentences from A. The *models* and the *derives* concepts are related, i.e., a derivation system needs to be *sound*, meaning that only entailed sentences can be derived. A desirable feature for a logical system is *completeness*. For a complete system, all entailed sentences can be derived. If a set of inference rules is both sound and complete, then entailment and derivation coincide.

In situation theory, the symbol, |=, stands for *supports*. This is a softer requirement than ``satisfies;" it admits some incompleteness of the relational structure. Situations in the world play the role of models (relational structures). Infons, on the other hand, are sentences, i.e., they are part of logical theories. Relations among infons can thus be modeled using the ``derives" relation. The model-theoretic meaning for ``derives" is provided by the ``satisfies" relation in the usual model-theoretic sense.

Our interpretation of situation theory can be discussed using Figure 4. The figure shows an oval representing the World and a rectangle representing an Agent. The Agent is connected with the World; this fact is captured by the arrows annotated with the label *supports*. The Agent represents information about the World in terms of infons. The figure indicates that at a given time t_1 and location l_1 a situation s takes place in the World. This is captured by the infon <<type, s, Situation, l_1, t_1>>. We used here the term ``type" that in OWL has the meaning of ``instance of" and the term ``Situation" to indicate that this is a class of situations.

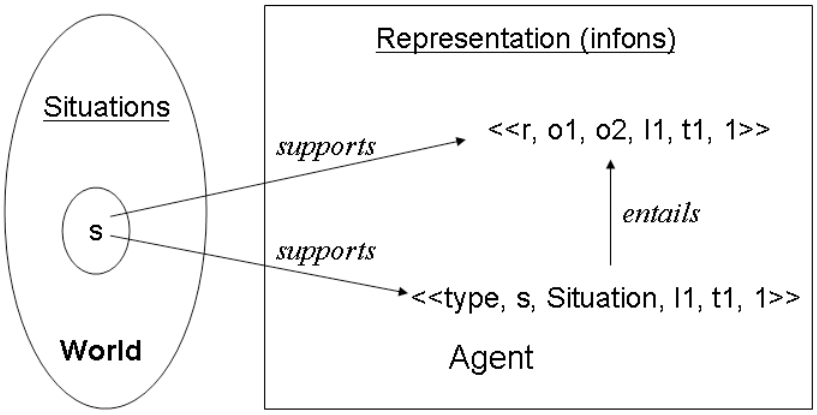


Figure 4. Agent's conncection to the world.

Since our Agent is a logical agent, it uses formal logic to derive facts about situations. In this particular case, the Agent *derives* that the relation r holds using logical inference. This fact is expressed by the *entails* relationship between two infons - from the fact that an infon about situation s holds, another infon can be entailed, i.e., one that captures the fact that in this situation, a relation r among objects o_1 and o_2 holds. The formalization of entailment is given by the standard model-theoretic semantics.

Therefore, in our framework, we capture the basic concepts of situation theory, i.e., the ``supports" relation, the partiality of the knowledge of real situations (it is a feature or imperfection of the Agent), the inference of facts from situations, and the fact that particular facts hold in a situation.

We next discuss the basic components of situation theory. This will be followed by a discussion of the situation theory ontology.

5.3.5 Objects and Types

The basic elements of situation theory are *objects* (also called *uniformities*) and *types* [16]. We start the presentation of our formalization of STO by showing the top-level structure of STO. In particular, we first explain how *objects* and *types* are interrelated within STO. In Figure 5 we show the pattern that is used throughout the construction of STO. The main idea is that the ontology has two *meta-levels*. The class TYP is the top-level class representing *types*. It has a number of subclasses (subtypes) as described in the next section. In Figure 5, we show only one subtype, IND. Instances of this class are classes. In this case, Individual is the class that is an instance of IND. In the figure, the relation of ``instance of" is represented by an arrow and a label, io. The class

RelevantIndividual is a subclass of Individual. The ``subclass" relation is represented by an arrow with the label, isa. In STO, each kind of object has two associated classes - the type of object class (analogous to IND) and a class that collects instances of a given type (analogous to Individual).

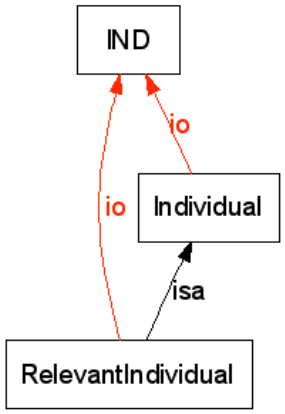


Figure 5. A fragment of STO - types.

For instance, consider the class called Dog. Instances of this class are representations of particular dogs. The class Dog is an instance of IND. We can also have subclasses of Dog such as YellowLab which contains descriptions of various dogs of this breed. The class YellowLab is an instance of IND.

By virtue of the fact that the STO is expressed in OWL, one can construct classes using the OWL class constructors. These constructors are sufficient for the most common forms of type construction in situation theory. When the OWL class constructors are insufficient for constructing a situation-theoretic type, one can use rules. Both the OWL constructs and the rules have fully specified and computer-interpretable semantics.

5.3.6 Basic Types

Now we discuss all the types of objects of situation semantics.

TYP – the class of types, as described in Section 5.3.6.

IND – the type of individuals. The corresponding class, Individual, is an instance of IND. These are entities perceived by an agent using its connections (as explained earlier).

In situation theory individuals are usually denoted as a, b, c, … In our ontology we provide a class called Individual. So particular individuals a, b, c, … are instances of this class. Examples of individuals are Rex (an instance of the class Dog), and Fluffy (an instance of the class Cat). In STO, these facts would be expressed in OWL. In mathematical notation, the facts about Rex and Fluffy would be presented by the unary predicate expressions Dog(Rex) and Cat(Fluffy).

RELn – the type of n-place relations. STO represents these using the Relation class. In situation theory, the first position of an infon specifies the relation of the infon. STO has a property relation from ElementaryInfon to Relation that serves the role of the first position of an infon. Elementary infons must satisfy the constraint of compatibility of arguments (see the discussion of parameters below).

Relations, denoted as P, Q, R, … are described by their names and the types of the objects allowed in particular attribute places. An example of a relation is chases. An example of a tuple of this relation is <Rex, Fluffy>. In STO, the tuple would be represented using a binary predicate, chases(Rex,Fluffy). The fact that this tuple is a member of the chases relation would be represented as relation(chases(Rex,Fluffy)) = chases.

ATTR – the type of attributes. Situation theory provides types specifically devoted to capturing locations (LOC) and time instants (TIM). In STO, these two types are subtypes of ATTR. To capture instances of locations and time instants, STO provides classes Location and Time, respectively. For other attributes STO provides the class Attribute, which is a superclass of both Location and Time. OWL has a rich collection of time notions as a result of its support for the XML Schema datatypes.

In situation theory, infons may include information about locations and time of occurrence of a particular situation. For instance, the situation in which the dog, Rex, is chasing the cat, Fluffy, can be expressed by the proposition

$$s \models \ll \text{chases, Rex,Fluffy, l1,t1, 1} \gg$$

The location of the situation is l1 and the time is t1.

VAL – the type of values. Situation theory does not have such a type. In situation awareness, however, one needs to speak of such values as 5m⁄s or 30km. In order to be able to model this kind of thing, STO provides the VAL type and the Value class. We model the type POL - polarity - of situation semantics as a subtype of VAL. Polarity represents the truth value of an infon; it can be either 0 or 1. In STO, polarity is represented using the Boolean data type of XML Schema.

DIM – the type of dimensions. This type captures dimensions (information about the systems of units) in terms of which particular values are expressed. Although situation theory does not provide such a type, it is a very important type for modeling various physical and other phenomena. Instances of this type are elements of the class Dimensionality. Examples of such instances are [m⁄s] and [km].

SIT – the type of situations. This type corresponds to the Situation class that has already been discussed. The semantics of the Situation class is the same as in Barwise, i.e., an abstract situation is the set of those infons that are supported by the same concrete situation s. According to situation theory, the word “the same” in this sentence means “whatever the agent perceives as the same.” In other words, the extent of this class is deferred to the capabilities of the agent. In our approach, the constraints placed on the

Situation class capture our understanding of the agent's perceptual capabilities, i.e., they capture what the agent perceives as "the same situation." We provide an example of a situation later in the paper, after we introduce the notions of "utterance situation," "focal situation" and "resource situation."

Situations in situation theory involve objects and relations among the objects. In order to model them, STO has the relevantObject and relevantRelation properties. We require that a situation must have at least one relevant object and at least one relevant relation. These kinds of constraint are expressed in STO as existential restrictions on the Situation class.

PAR – the type of parameters. Situation theory uses parameters as a mechanism for constructing types. Parameters are used in infons; they serve the same role as variables in rule-based systems and languages. Because STO is expressed in OWL, it has a rich set of mechanisms for class construction that does not rely on variables. As discussed above, these mechanisms are sufficient for most class constructions. When these mechanisms are not adequate, one can specify a class using rules. Rules use variables in much the same way that situation theory uses parameters.

Situation theory provides the notion of restricted parameters to allow one to restrict the ranges of parameters used in an infon. STO has properties par1Type, par2Type, ..., whose domain is Relation and the range is a (restricted) type, that allow one to restrict the slots of a relation to specified classes.

Parameters of situation theory are variables which can be set to specific object values. The setting of one or more parameters to specific objects is done by means of an anchor which maps the variables to the desired values. We will say that an infon is anchored when all of its parameters have been anchored to objects. STO uses the properties anchor1, anchor2, ..., whose domain is ElementaryInfon and whose range is Object, to specify the object values in the slots of an instance of ElementaryInfon. The instances of ElementaryInfon that can be part of a given relation will be restricted exactly as in situation theory, i.e., the type of the value to which a triple is anchored to must be exactly the same as the type of the slot of the relation that the tuple is part of. We require that both par<n>Type and anchor<n> properties be functional.

In situation theory the parameters representing individuals, situations, location or time are denoted by ȧ, ṡ, l̇ , ṫ, respectively. They can be of any type of object that is part of situation theory. An example of a parameter could be FourLeggedAnimal. In that case the relation chases could be restricted to consider only four-legged animals that chase each other. In STO, a class FourLeggedAnimal would be constructed as a subclass of Individual and restricted to those instances of Individual for which the property numberOfLegs has the value 4.

INF – the type of infons. In situation theory, this includes both elementary and compound infons. We introduce a class ElementaryInfon for elementary infons, and we use OWL class constructors and rules to deal with compound infons. For example, consider the general form of an elementary infon:

$$\ll R,a1,...,an,\ 0/1 \gg$$

We can gain an understanding of what is possible to represent in STO by considering all possible fillers for particular slots in the above representation of an infon.

The first slot, R, can be filled with a representation of a relation. In STO, this is an instance of the class Relation. Since STO is expressed in OWL, any OWL property can also fill this slot. Such a property is always binary as an OWL property, but in STO, it can have additional slots, such as the time when the property holds for two individuals. We have already seen an OWL property, type, in this role.

The slots a1,…,an can be filled with: individuals, relations, location (spatial and temporal), situations, and types of all of the above. Compound infons can be expressed using OWL expressions as well as rules. Examples of these will be discussed later.

5.3.7 Reasoning Within Situation Semantics Theory

In situation theory, an agent reasons by applying knowledge that is expressed with constraints. Constraints link situation types. A constraint that situation type S1 is linked to another situation type S0 is written as

$$S0 \Rightarrow S1,$$

and one says that S0 involves S1. The meaning of such a constraint is that whenever a situation s0 is an instance of S0, there is a situation s1 that is an instance of S1. When s1 is the same situation as s0, i.e., s1 = s0, Devlin refers to the involvement as being reflexive. Constraints play the role of laws of the world, e.g., physical laws. In the STO, the reflexive constraints correspond to subclass relationships among situation types. In other words, to express that S0 reflexively involves S1 one specifies that S0 is a subclass of S1. A subclass relationship is a special kind of rule which specifies that if a situation belongs to one situation type, then it also belongs to another one. Reasoning using such rules is called subsumption, and it is the basis for description logic [29], which is the underlying logic for OWL. While subclass relationships are a “built-in” feature of OWL, there is no relationship in OWL corresponding to non-reflexive involvement, although either kind of involvement can be expressed using rules.

5.3.8 The Situation Theory Ontology

Now we show how situation theory is formalized as an ontology; we call it the Situation Theory Ontology (STO). A graphical representation of STO is shown in Figure 6 and Figure 7. Since STO is a relatively complex ontology, in these figures we show only some of the classes in partial views.

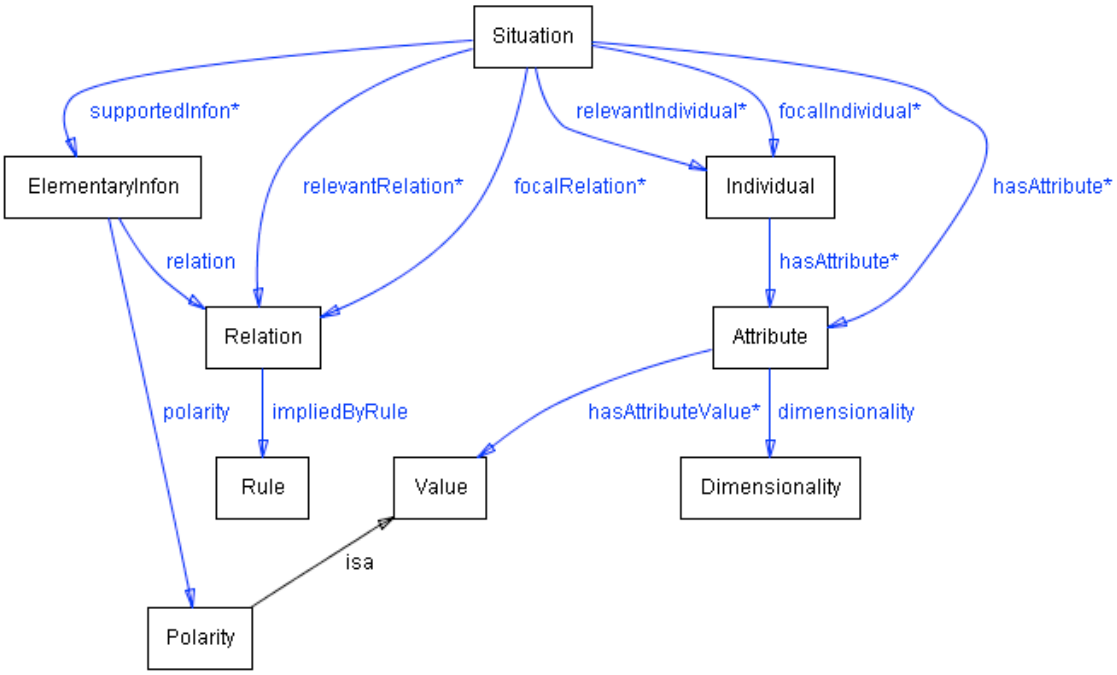


Figure 6. Main classes and properties of STO

The ontology is visually represented using a Protégé plugin called OntoVIZ. The boxes in this notation represent classes. A class is interpreted as a set of instances that satisfy all the constraints and restrictions associated with the class. The rectangles show class names. Arrows represent properties. Names of properties appear as labels on the arrows. In OWL, properties are binary relations. The class at the tail of an arrow is the domain of the relation and the class at the head of the class is the range of the relation. The complete ontology is available at http://vistology.com/ont/2006/STO/STO.owl.

Situation is the central class. Instances of this class are specific situations. This class is a direct counterpart of the abstract situation concept in situation theory. The second class is the Individual class, which is a counterpart of the individuals in situation theory. Similarly, Relation captures the n-ary relations. In order to provide a means for inferring relations we introduce the class Rule. Instances of this class capture axioms of the domain that can be used for inferring whether a given relation holds in a situation or not. Attribute is a generalization of locations and time instants in situation theory. Instances of this class are attributes of individuals and situations. An attribute may have a dimension associated with it (e.g., [m⁄s] or [m^2]. For this purpose, we introduce the class Dimensionality. We also introduce the class Polarity. This class has only two instances that correspond to the two possible values associated with a tuple, either that a given tuple holds or that it does not hold. In situation theory these polarity values are denoted as '1' and '0'. The fact that polarity is a special case of Value is specified in OWL using the subClassOf property. In the OntoVIZ notation this is shown by the isa label.

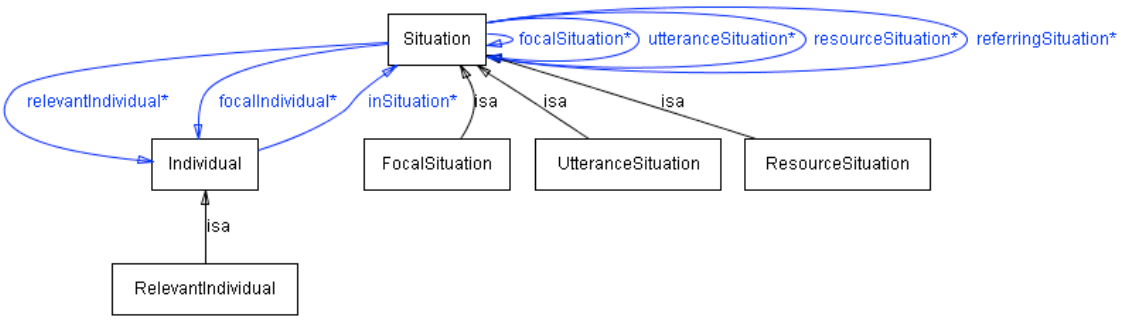


Figure 7. Situation Types (partial view)

Classes of STO are related through a number of OWL properties. Situations are linked with four kinds of entities. First, the property relevantIndividual captures the individuals that participate in a situation. The property relevantRelation is used to assert that a given kind of relation is relevant to a given situation. Since situations are objects, they can have attributes of their own. Attributes of situations are captured by the hasAttribute property. The domain of this property also includes Individual. Attributes have dimensionality as well as hasAttributeValue properties.

5.3.9 Subclasses of Situation

Situation semantics is a study concerned with utterances. In situation semantics, three kinds of situations are distinguished: utterance situation, resource situation and focal situation. Consequently, in our ontology, we provide three subclasses of the class Situation: UtteranceSituation, ResourceSituation and FocalSituation (see Figure 7). These are provided primarily to show that the STO is rich enough to express Barwise's situation theory. If ones primary concern is not with utterances, then one can introduce other subclasses of Situation.

In situation theory, the meaning is acquired through the speaker's connections (links) to a situation. To be compatible with Devlin, these links are captured in the STO by means of the focalIndividual and focalRelation properties.

The most important type of situation to Devlin is the utterance situation. "This is the context in which the utterance is made and received." In situation theory, an utterance is represented as an expression, Φ, in some language, e.g., in natural language. In order to acquire the meaning, this expression needs to be linked to a real situation, s, and represented as a (typically compound) infon. The links between Φ, s and the infon are in the so called speaker's connections.

In our formalization of situation theory we interpret utterances as queries which come from the user of a formal situation awareness system. Queries thus partially provide the "speaker's connection". In our SAW Core Ontology [24] we used to call this class Goal. It is like a perspective that gives focus to what should be considered as relevant for a specific situation.

In some cases, an utterance refers to another situation, i.e., another situation is used in a support role. The kind of situation that an utterance situation is referring to is called a resource situation. It is a situation that is used as a kind of background for reasoning with the current situation. And finally, a focal situation (or described situation) is that part of the world that is relevant to a given utterance. Because our emphasis is on understanding a situation based on sensory data, the focal situation is the most important type of situation. As Devlin explains, "Also known as the described situation, the focal situation is that part of the world the utterance is about. Features of the utterance situation serve to identify the focal situation."

We introduce two pairs of properties that are inverses of each other (inverses are denoted by '↔') whose domain and range is Situation:

- The properties focalSituation↔utteranceSituation are used to link instances of UtteranceSituation with instances of FocalSituation.
- The properties resourceSituation ↔ referringSituation are used to link instances of UtteranceSituation with instances of ResourceSituation.

Moreover, we introduce the relevantIndividual property whose domain is Situation and range is Individual along with its inverse inSituation. These properties capture links between situations and individuals that are part of the situation in one direction, and individuals that are in a situation, respectively. focalIndividual is a subproperty of relevantIndividual.

In Figure 7 we show the three situation types, and a partial view of the properties of STO that are related to the three situation types as well as to the notion of relevant individual. An instance of UtteranceSituation is required to have at least one focalIndividual (i.e., it must have a value for property focalIndividual) and at least one focalSituation. An instance of FocalSituation is required to be associated with an UtteranceSituation. A situation is qualified as a ResourceSituation only if it is related to another situation by the referringSituation property.

We interpret Devlin's discussion about focal situations to be that focalIndividual is a subproperty of relevantIndividual; in other words, all focal individuals of a situation are also relevant. Our intentions go one step further - in applications of the ontology based approach to situation awareness we would like to be able to derive all of the other individuals that are relevant to a situation. This derivation is based on the knowledge of focal individuals, as well as other knowledge that can be extracted either from a query or from background knowledge.

5.3.10 Example

In this section we present some examples of situations. First we present their descriptions in natural language. In our discussion we will refer to Figure 8, a World that gives rise to a number of situations. This figure shows some objects involved in these situations. One

of these is that the dog, Rex, is chasing the cat, Fluffy. Jim is the owner of Fluffy and thus is involved in watching the chase due to his concern about Fluffy's well being, if chasing means that Rex is threatening Fluffy, but not if Rex is just playing with Fluffy. There are also two other objects, a mouse (Mickey) and a flower (we call it Tulip1), which are not involved in the chasing or watching situations but which might be involved in some other situations. Nevertheless, the chasing situation is of interest to Mickey, since it knows that it is safe from the cat for as long as the cat is being threatened by a dog. Now we list some of the situations related to the World shown in Figure 8 that are of interest to us in this paper.

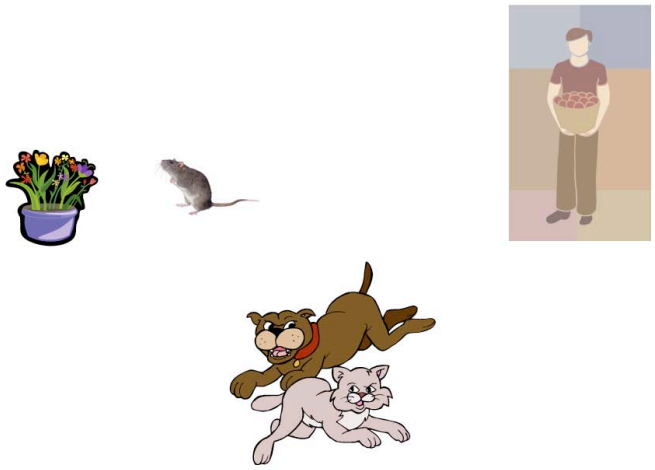


Figure 8. An example of a World

RexThreatenFluffy. In this situation, Rex chases Fluffy. Relevant relations include chases and threaten. Both are binary relations. The first parameter represents the chaser, or threatener, and the second the one being chased, or the victim. Attributes of this situation may include the location of the situation and the time associated with the occurrence of the chase. Objects may have attributes of their own, like location or velocity. The situation may include an attribute of 'rate of approach', i.e., the difference between the velocities of the pursuer and the pursued.

RexPlayingWithFluffy. This is similar to the previous situation except that playWith rather than threaten is relevant. One can distinguish this situation from the previous only if one knows whether the chaser is friendly toward the other animal.

JimWatching. Jim, the owner of Fluffy, is involved in this situation. He is a focal object of this situation and so is Fluffy. The relevant relations include the watch relation in which the watcher is the first argument and the object being watched is the second. Rex is in this situation too, although his belonging to this situation comes through the fact that Jim is interested in the RexThreatenFluffy situation because Fluffy is involved in that situation. If another dog were threatening Fluffy, that dog would be in this situation. The

RexThreatenFluffy situation plays the role of a resource situation for the JimWatching situation.

SafeFromCat. From the point of view of Mickey (the mouse), safety from a cat is one of the main concerns. The relevant relation is safeFromCat with the first argument being the subject (in this case Mickey) and second argument being the object (in this case Fluffy). The RexThreatenFluffy situation again plays the role of a resource situation. Since Fluffy is preoccupied with the possibility of being attacked, Mickey is safe from the cat.

5.3.10.1 Representation of Situations in STO

In this section we describe how the situations discussed above are captured in a formal language.

First, in order to distinguish general terminology for the STO from the specific terminology of the examples, we introduce two namespaces. The sto: prefix will be used for distinguishing terminology in the STO namespace (i.e., tt http://vistology.com/ont/2006/STO/STO.owl). The rex: prefix will be used for terminology specific to the examples (i.e., tt http://vistology.com/ont/2006/STO/Rex.owl).

Second, we extend STO to accommodate the specifics of the world and of the situations. Towards this aim, a number of classes and relations are introduced. The classes include Person, Cat, Dog, Mouse, and Flower. These classes are subclasses of Individual.

Third, we assert a number of facts about the world (shown in Figure 8) in terms of the extended ontology. In particular, we assert that Jim, Fluffy, Rex, Mickey, and Tulip1 be instances of Person, Cat, Dog, Mouse, and Flower, respectively. Each of these individuals have attributes of Location. In our example Jim has location L_Jim, Fluffy has location L_Fluffy and so on. Each of the attributes has a dimensionality. All attributes of type Location in our representation have dimensionality [m] (for simplicity), and all velocity attributes have dimensionality [mps]. Additionally, we introduce relations watch, chases, and safeFromCat with arguments of appropriate types. Using the usual mathematical notation, the arguments of these relations are as in watch(Person,Cat), chases(Dog,Cat), and safeFromCat(Mouse,Cat). All these relations are defined as instances of Relation.

Having done this, we can now discuss various situations that can be considered in this world. Each of the situations starts with an utterance situation. Since we are not going to deal with natural language processing in this paper, and moreover, some of the agents in this world don’t use natural language at all, we will focus on the utterance situations rather than the utterances that give rise to them.

5.3.10.2 Representation of `RexThreatenFluffy’

In this situation, Rex and Fluffy are both moving at high speed in the apartment. This is shown by the location and velocity infons. It is also known that Rex is not friendly

toward Fluffy. These are called the observed infons because they are determined by sensors or some other source of information. They are also called asserted infons because they must be explicitly given to the rule engine. The other infons that are supported by the situation are derived from the observed infons or from other derived infons. For example, one can use the locations and velocities of Rex and Fluffy to conclude that Rex is moving in the direction of Fluffy. Similarly, one can infer that Rex is near Fluffy. The fact that Rex is chasing Fluffy is derived from other derived infons, and the fact that Rex is threatening Fluffy is derived from both a derived infon and an observed infon. Mathematically the last three infons just mentioned are the relation tuples near(Rex,Fluffy), chases(Rex,Fluffy) and threaten(Rex,Fluffy). The infons mentioned above are written as follows in the notation of Situation Theory:

RexThreatenFluffy |= ≪ location, Rex,LxRex, 1 ≫
RexThreatenFluffy |= ≪ velocity, Rex,VxRex, 1 ≫
RexThreatenFluffy |= ≪ location, Fluffy, LxFluffy, 1 ≫
RexThreatenFluffy |= ≪ velocity, Fluffy, VxFluffy, 1 ≫
RexThreatenFluffy |= ≪ near, Rex,Fluffy, 1 ≫
RexThreatenFluffy |= ≪ chases, Rex,Fluffy, 1 ≫
RexThreatenFluffy |= ≪ isFriendlyToward, Rex,Fluffy, 0 ≫
RexThreatenFluffy |= ≪ threaten, Rex,Fluffy, 1 ≫

To illustrate what these infons would look like when written in the OWL Abstract Notation, we show the last two infons as follows:

```
Individual(sto:RexThreatenFluffy
  type(sto:ResourceSituation)
  value(sto:supportedInfon Individual(_ type(sto:ElementaryInfon)
    value(sto:anchor1 rex:Rex) value(sto:relation rex:isFriendlyToward)
    value(sto:anchor2 rex:Fluffy) value(sto:polarity sto:_0)))
  value(sto:supportedInfon Individual(_ type(sto:ElementaryInfon)
    value(sto:anchor1 rex:Rex) value(sto:relation rex:threaten)
    value(sto:anchor2 rex:Fluffy) value(sto:polarity sto:_1))))
```

The main difference between situation theory notation and OWL notation is that the implicit features in situation theory notation are explicit in OWL notation. For example, the double angle brackets in situation theory are explicitly labeled with type(sto:ElementaryInfon) in OWL notation. Similarly, the meaning of one of the slots in situation theory notation is implicitly determined by its position in the infon, while in OWL notation the purpose of each slot is explicitly labeled, and the slots may appear in any order. For example, the first slot of an infon in situation theory notation is the name of the relation, while in the OWL notation it is explicitly labeled as in value(sto:relation sto:threaten) and does not have to occur first in the infon.

Yet another representation of these infons is shown in Figure 9 using OntoVIZ. Because infons have no explicit labels, Protégé introduced labels for them (i.e., @_:A21 and @_:A26).

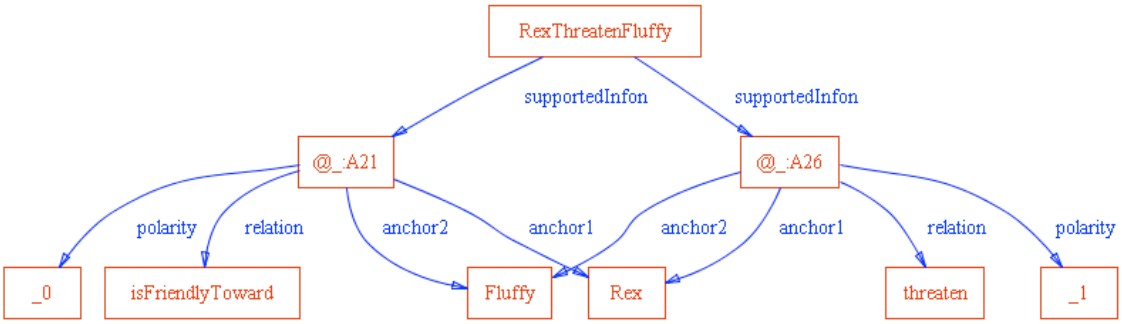


Figure 9. Two infons of the RexThreatenFluffy situation

5.3.10.3 Representation of `RexPlayingWithFluffy'

The Playing situation is nearly the same as the Threaten situation, except that Rex is friendly toward Fluffy, and thus the inferences are different. Relevant derived infons in the situation include that Rex is near Fluffy, Rex is chasing Fluffy and Rex is playing with Fluffy. Mathematically these three relations are near(Rex,Fluffy), chases(Rex,Fluffy) and playingWith(Rex,Fluffy). The infons for this situation are the same as the infons for the RexThreatenFluffy situation except for these:
RexPlayingWithFluffy |= ≪ isFriendlyToward, Rex, Fluffy, 1 ≫
RexPlayingWithFluffy |= ≪ playWith, Rex,Fluffy, 1 ≫

5.3.10.4 Representation of `JimWatching'

In the JimWatching situation we presume that the background resource situation is the RexThreatenFluffy situation. Most of the infons in this situation are derived from the background resource situation. The additional infon for this situation is the following:
JimWatching |= ≪ inDanger, Fluffy, 1 ≫
In other words, Jim concludes that his cat Fluffy is in danger.

5.3.10.5 Representation of `SafeFromCat'

The SafeFromCat situation is similar to the JimWatching situation, but it is from the point of view of Mickey who comes to a different conclusion:
SafeFromCat |= ≪ safeFromCat, Mickey, Fluffy, 1 ≫
In other words, Mickey concludes that it is safe from the cat.

5.3.11 Using Formal Representations

In this section we discuss some of the possible uses of formal representations of situations and advantages from such an ontology-based approach.

One of the great advantages of having situations represented in a formal language is that facts that are not explicitly stated can be derived using an inference engine. In this section we consider how one derives facts within a single situation, given that some other facts are known. Inference in OWL includes a form of reasoning called subsumption reasoning that it is based on subclass relationships. We would like to express the rule that a dog that is chasing a cat and that is also unfriendly with the cat is threatening the cat. More specifically, we would like to state this rule in the case of Rex. To express the rule using subsumption one uses classes for each of the three parts of the rule. Inference can be invoked when such classes are defined and facts about instances of the classes are known. In this particular case, it is assumed that the following classes are already defined in the ontology:

1. RexUnfriendlyCat, the class of cats with which Rex is unfriendly;
2. RexChasingCat, the class of cats that Rex is chasing; and
3. RexThreatenCat, the class of cats that Rex is threatening.

The three definitions are shown at http://vistology.com/ont/2006/STO/Rex.owl.

As the definitions are similar to one another we explain just the first one. To define this class, we first define another class, called RexUnfriendlyInfon, consisting of the elementary infons that express the fact, in the situation, that Rex is unfriendly toward a cat. The class of such infons is the intersection of five classes:

1. The class of elementary infons, ElementaryInfon.
2. The class of infons supported by the RexThreatenFluffy situation. Mathematically, this is the set {x|supportedBySituation(x,RexThreatenFluffy)}.
3. The class of infons for which the relation is isFriendlyToward, or {x| relation (x, isFriendlyToward)}.
4. The class of infons for which the first slot of the isFriendlyToward relation has value Rex, or {x|anchor1(x,Rex)}.
5. The class of infons for which the polarity is 0, or {x|polarity(x,0)}.

In Situation Theory this class of infons is written:
RexThreatenFluffy |= ≪ isFriendlyToward, Rex,c,0 ≫
where c is a parameter that can have any cat as its value. This is expressed in OWL by leaving the second slot of the isFriendlyToward relation unconstrained in the definition of the class RexUnfriendlyInfon.

The class RexUnfriendlyCat is defined as the intersection of two classes:

1. The class of cats, Cat.
2. The class of objects which are in the second slot (i.e., anchor2) of an infon in RexUnfriendlyInfon. Mathematically, this is the set {x|∃i ∈RexUnfriendlyInfon s.t. anchor2(i) = x}. To express this in OWL, one must use the relation anchor2inverse which is the inverse of anchor2, i.e., the property for which the subject and object

have been reversed. In other words, one must express the set mathematically as {x|anchor2inverse(x) ∩RexUnfriendlyInfon≠ ∅}.

SubClassOf(intersectionOf (sto:RexUnfriendlyCatsto:RexChasingCat)sto:RexThreatenCat)

Figure 10 Subsumption rule for threatening

Once one has defined the three classes, one can represent the rule by asserting that the third class is a subclass of the intersection of the first two as shown above. In other words, if a cat is both in RexUnfriendlyCat and in RexChasingCat, then this cat is in RexThreatenCat.

In the RexThreatenFluffy situation, the infon ≪ isFriendlyToward, Rex,Fluffy, 0 ≫ was asserted. This infon can be expressed mathematically by stating that Fluffy∈RexUnfriendlyCat. In OWL/XML it is written as: <RexUnfriendlyCat rdf:about="#Fluffy"/>

The infon ≪ chases, Rex, Fluffy, 1 ≫ was derived by rules similar to the one explained here, the result being that Fluffy∈RexChasingCat. The subsumption rule above then allows one to conclude that Fluffy∈RexThreatenCat. Stated in terms of Situation Theory, this is the infon: ≪ threaten, Rex, Fluffy, 1 ≫

The advantage of the form of inference described in this section is that it is situation-specific. Different situations will not only have different infons, but also different inference rules. This can be very useful when customized situation-specific rules are needed. However, many rules apply to more than one situation, and so it is useful to have a mechanism for stating rules that have more general applicability. This is the subject of Section 5.3.13.

5.3.12 Inference Using Rules

While the above example was accomplished completely within the scope of OWL, there are many cases for which OWL is not sufficiently expressive to capture all of the desired concepts. In particular, it is not possible to construct a complex property defined as the composition of other properties. This limitation derives from OWL's lack of variables and the inability to define joins [30]. Consider for example the notion of chasing. As discussed in Section 5.3.10.2, this can be inferred from the relations near and inDirectionOf. The following simple Horn clause rule expresses this inference for arbitrary objects:

If
  near(X,Y) and inDirectionOf(X,Y)
Then
  chases(X,Y)

This rule has two advantages over the rule in Figure 10. First, it is more general, applying to arbitrary dogs, not just Rex. Second, it is much more succinct. To give another example of the power of rules, consider another way that chasing might be inferred. If a species preys on another species, one can infer that a particular animal of a predator species will chase a particular animal of the prey species if the former animal sees the latter animal. Using a Horn clause this can be expressed as follows:

If
belongsToSpecies(X,S) and belongsToSpecies(Y,T)
and preysOn(S,T) and sees(X,Y)
Then
chases(X,Y)

The two Horn clause rules above have been implemented in a rule language supported by the BaseVISor inference engine [31]. BaseVISor is optimized for the processing of RDF/OWL triples and implements the semantics for RDF/RDFS and a subset of OWL called R-Entailment [32]. When the chase rule is submitted to BaseVISor along with the facts from the RexChasesFluffy situation and the additional fact that Rex "sees" Fluffy, the fact that Rex chases Fluffy is automatically derived and added to the collection of facts about the situation. In addition to deriving this fact, BaseVISor also infers other facts about the situation using the RDF/OWL semantics defined by the R-Entailment axioms; one such inferred fact is that the situation is an instance of the class RexChasesFluffySituation discussed in the next section.

5.3.13 Inferring Situation Types

One of the important types of inference about situations is the classification of a given situation to a situation type. This form of class definition differs from that in Section 5.3.11 in that it classifies situations where some behavior has occurred, rather than classifies objects within a situation.

Consider, for example, the notion of chasing. In any situation, a dog is chasing a cat if the dog is near the cat and moving in the direction of the cat. We would like to express this as a subsumption rule. As we are specifically interested in Rex and Fluffy, we will express the rule in terms of them. Mathematically, this rule can be expressed as a subset relation between classes of situations:
$S1 \cap S2 \subseteq S3$
where
$S1 = \{s|s \models \ll near, Rex,Fluffy, 1 \gg \}$
$S2 = \{s|s \models \ll inDirectionOf, Rex,Fluffy, 1 \gg \}$
$S3 = \{s|s \models \ll chases, Rex,Fluffy, 1 \gg \}$

```
<sto:SIT>
   <owl:intersectionOf rdf:parseType="Collection">
     <sto:SIT rdf:about="#RexNearFluffySituation"/>
     <sto:SIT rdf:about="#RexInDirectionOfFluffySituation"/>
```

```
  </owl:intersectionOf>
  <rdfs:subClassOf>
   <sto:SIT rdf:about="#RexChasesFluffySituation"/>
  </rdfs:subClassOf>
 </sto:SIT>
```

In situation theory this relationship between the situation types would actually be written S1 ∩ S2 ⇒ S3, and the relationship between the situation types is referred to as reflexive involvement as discussed in Section 5.3.7. In the STO, the situation types are shown in http://vistology.com/ont/2006/STO/Rex.owl, where they are called RexNearFluffySituation, RexInDirectionOfFluffySituation, and RexChasesFluffySituation, respectively. The XML above shows how the rule is written in OWL/XML.

5.4 Querying Formal Representations

After one has expressed all of the rules that apply in general for any situation and also the ones that apply for either just the particular situation or a type of situation, one can ask questions about the situation by using queries. The difference between queries to a database and queries to an OWL knowledge base is that the answer to a knowledge base query may include facts that are inferred as well as facts that have been explicitly asserted. We now give some examples of queries that may be used in the situations we have described above. We use the SparQL language [33] to express these queries.

Consider first how Jim would determine whether Fluffy is being threatened by Rex. The query for this is:

```
BASE      <http://example.org/jwf/>
PREFIX    sto: <http://vistology.com/onto/STO/>
ASK
FROM NAMED <http://example.org/jwf/>
WHERE     { GRAPH <http://example.org/jwf/>
          { ?infon sto:relation <threaten>;
                   sto:anchor1 <Rex>;
                   sto:anchor2 <Fluffy>;
                   sto:polarity sto:_1 .
          }
         }
```

The BASE is the URI for the unqualified terms in the query. These are the ones in the specific situation to be queried. The PREFIX is the prefix for terms in the STO. The FROM NAMED clause specifies the source for the data to be queried; namely, the facts in the situation. In this case, only one situation is being considered, but queries can involve several situations at the same time. The WHERE clause specifies a graph pattern that is matched against the knowledge base. ASK signifies that the result of the query is only whether there is at least one match, not what the matches might be.

Similarly, Mickey can query whether it is safe from the cat. A somewhat more complicated query is a request for all solutions to the pattern rather than just whether there is a solution. For example, Jim might ask which animals are chasing each other in the following query:

```
BASE      <http://example.org/jwf/>
PREFIX    sto: <http://vistology.com/onto/STO/>
SELECT    ?dog ?cat
FROM NAMED <http://example.org/jwf/>
WHERE     { GRAPH <http://example.org/jwf/>
          { ?infon sto:relation <chases>;
                   sto:anchor1 ?dog;
                   sto:anchor2 ?cat;
                   sto:polarity sto:_1 .
          }
         }
```

5.4.1 Communicating Situations

Once a situation is captured in a formal language, it can be communicated to another agent and the receiving agent will interpret the situation in exactly the same way as the sending agent. In the JimWatching situation, Jim must be able to distinguish whether Rex is threatening or playing. If he has been told by the neighbor who owns Rex that Rex is friendly toward Fluffy, then Jim would not infer that Fluffy is in danger. This information could be conveyed to Jim as the infon:

JimWatching |= ≪ isFriendlyToward, Rex, Fluffy, 1 ≫

In STO this could be expressed as follows:

```
<RexFriendlyCat rdf:about="#Fluffy"/>
```

provided the definition of RexFriendlyCat is available in the JimWatching situation.

Alternatively, and more in the spirit of situation theory, the information could be conveyed using situation types. Namely, there was a situation in which Rex and Fluffy were previously observed to be playing, with no harm to either. From this one can infer that Rex is friendly toward Fluffy.

This may seem to be a lot of information to exchange, but if the two agents have already exchanged information about the situation and some new piece of relevant information about the situation becomes available to one of the agents, the agent can send only this new piece to the other agent with a reference to the resource representing the situation. For example, Jim could communicate with his neighbor about the current situation, and the neighbor would add useful information which would enable Jim to determine whether Fluffy is being threatened. For such a communication to be useful, Jim and his neighbor must share a common understanding about dogs, cats, and their behavior toward one another, in general, as well as about Rex and Fluffy in particular.

5.4.2 Further Extensions of STO

The main contribution of the work described in this section is to provide a computer-processable semantics for situation theory which is compatible both with the situation theory of Barwise and with Endsley's model of human situation awareness. To achieve this we expressed situation theory as a formal ontology in OWL. The advantage of an ontology based approach to situation awareness is that once facts about the world are stated in terms of the ontology, other facts can be inferred using an inference engine. This is particularly important for situation awareness since it heavily relies on the knowledge of relations. Because there are so many possible relations, it is impractical to expect that procedures could be written for all potential relations.

In this work we introduced a formalization of the basic components of a situation awareness ontology (STO). In particular, we formalized all the basic types, classes and relations of situation theory. We have also introduced a number of new types, classes and relations. It is our hope that the STO will be used as a starting point for the information fusion community to achieve consensus on an ontology for situation awareness. We hope that STO can play the role of a basis for a unifying theory of computer-based situation awareness.

In the future we plan to give more examples of the use of the STO in practice. We are also developing tools that can make it easier for an end user to use situation theory. However, the most important task is to develop a comprehensive situation awareness ontology through a community-wide effort so that the developed ontology can become a de facto standard ontology. In the following subsection we describe an extension to STO, called STO-B, i.e., an ontology for behavioral situations.

5.4.3 Behavioral Situation Theory Ontology

As mentioned above, the Situation Theory Ontology (STO) is an ontological reflection of the Situation Theory developed by Barwise and Perry and later by Devlin. It is particularly useful for modeling static situations. In order to be able to capture dynamic situations we had to extend it by adding some classes and relations to STO. The resulting ontology (STO-B) is shown in Figure 11.

As in STO, Situation is the central class. Instances of this class represent specific situations. ElementaryInfon is the class whose main role in this ontology is to capture the focus of attention of a situation. A situation object can have relationships with other instances of class Individual, capturing the participants in a situation. Situations also satisfy some relations – instances of class Relation. Situations are first-class objects and thus can have Attributes of their own, in additional to the attributes of the participants.

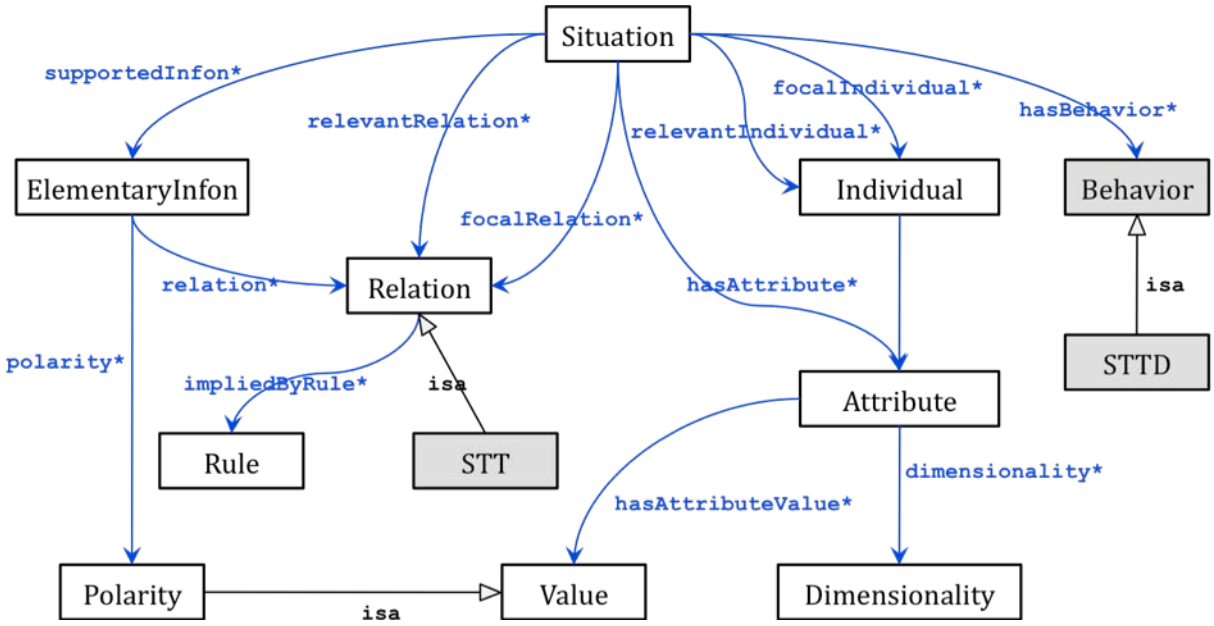


Figure 11. Situation Theory Ontology - Behavioral (STO-B)

In order to represent behaviors, the Behavior class has been added to STO. Typically, a behavior is represented by a State Transition Diagram (STD). However, since we are dealing with behaviors of situations it is necessary to define state of a situation. We suggest that Situation Type play this role. Consequently, we have a Situation Type Transition Diagram (STTD) in our ontology. The transitions themselves are instances of the class STT (for Situation Type Transition). Each such transition links two situation types. Thus the STT class is a sub-class of Relation. Instances of this class capture transitions between situation types, some of them occurring naturally (e.g. acknowledged by human agents) and other ones being implied by one or more rules. Our scenarios demonstrate rule based transitions.

6 Overview of Existing Approaches to Behavior Modeling

This section reviews some existing approaches related to the problem, including previous work performed at VIStology.

6.1 Formalization of States and Specifications

Before we arrived at our solution, we looked in the literature for several approaches. We analyzed in detail the Information Flow Theory of Barwise and Seligman. We also looked carefully at the coalgebras approach of Jacobs [34] and Harel's State Charts [35].

Our search quickly focused on the theory of Evolving Specifications based on a paper [70] from Kestrel Institute. The advantages of this model are its generality and incorporation of the dynamic aspect.

Let us quickly discuss how each of these approaches matched with our needs. Then we describe how our proposed solution deals with these problems.

The Information Flow theory [36,37] has some very interesting aspects for us. The notion of information morphisms seemed useful. That approach, however, focuses on not so much the dynamical aspect, but a possibility of inference in a complex theory from the knowledge of its parts that might be simpler to analyze.

The way the Information Channels are defined in [36] involves infomorphisms fi : Ai → S from some classifications Ai to a classification C. We think of Ai's as of some partial theories, and C is the whole theory. This is different from our goal of having a state machine where at every vertex we have a specification and the state transitions are guarded specifications. For example we allow loops, and multiple transitions from one state to another.

The approach from [36] might be very useful for us later on, but our stressing the possibility of implementing dynamic aspect makes it impossible to adapt Information Flow approach directly. Also, there is a simplifying assumption that we deal with a fixed set of "tokens" from which we take models of our theories.

The coalgebras approach has some attractive features suitable for situation theory. The main ones are the notions of bisimulations and invariants. Roughly speaking the bisimulations correspond to equivalence relations on the states of a state machine, satisfying some compatibility conditions. This corresponds precisely to what the situation type should be in our approach; a situation type will be an equivalence relation on the set of all instances of situations, satisfying the compatibility conditions with respect to events. The invariants are the properties of states that are preserved after applying new events. There were however some assumptions in the coalgebras approach that made us choosing Especs, namely some simplifying assumptions (polynomiality of the defining functor F, etc.). These assumptions are not satisfied even in the simplest scenarios we have in mind.

The state charts approach is quite useful, it is similar to the language we use : OWL. However there is a difficulty in adopting this approach directly, as there is no possibility of defining morphisms between two situations.

After going through these discussions we settled on an approach that is a compromise of the approaches cited above. A situation type scenario is an Espec together with a binary relations on the nodes of the graph. The Specs at the nodes of the graph are situations, the morphisms or guarded specifications that correspond to arrows are the events. The binary relation on the nodes divides situations into situation types, and a similar relation on the arrows, divides the events into event types. These relations have to be compatible in the sense that to events of the same type have to take two situations of a given type into situations of the same type.

6.2 Ontology Merging

In situation tracking, the incoming event can modify the situation. In such cases, the event and situation are somehow merged to represent the evolved situation. The more general problem is the process of automatically merging two or more ontologies into one consistent ontology. This is an ongoing challenge in the ontology integration community, Several applications allow graphical editing and offer suggestions of possible merges of classes, instances and properties but still require the approval of the human agent. These applications are more appropriate for aligning ontologies which may have different hierarchy structures.

Hovy [38] was one of the pioneers for using heuristics for ontology alignment and merging. The algorithm includes three types of heuristics which are used to calculate scores. Heuristics include string comparisons of names and definitions, semantic distances of concepts with common parents for ontologies with different hierarchies, and domain and range specifications of properties. If two concepts in different ontologies have a score above a certain threshold, the system suggests a merge and allows a user to accept or reject the recommendation. The consistency of the merged ontology is checked. If there are conflicts, the user is notified and must resolve the conflicts. Then the process is repeated until no more merges are performed.

Older tools that help automate ontology alignment and merging are Chimaera and PROMPT. Chimaera [39] suggests possible merges using heuristics of syntactic similarity and structural similarity. Structural heuristics involve analyzing classes from different ontologies that may be the same through subsumption, disjointness or properties related to the classes. Chimaera also identifies terms that may refer to the same concept if there were slight changes to constraints of acceptable property values. In addition, Chimaera can perform fairly simple checks inconsistencies in the merged ontology, such as name conflicts and dangling references.

PROMPT [40] is a plug-in for older versions of Protégé, a tool for creating and managing OWL ontologies [41]. It can either determine the mappings to align two ontologies or create a new merged ontology. PROMPT uses heuristics based on linguistic similarities to form suggestions [42]. Syntactic heuristics include synonymy, shared substrings, common prefixes and common suffixes. PROMPT also considers semantic similarities and differences based on the taxonomy of the ontologies. Like most merging tools, the user then has the option of accepting the change. The resulting ontology is checked for consistency using inference, including finding any redundancies in the taxonomy resulting from the merge and detecting violations of domain or range constraints inherited from superclasses. PROMPT then recommends possible solutions to the conflicts, which are ultimately resolved by the user.

OntoMorph is a system that can translate ontologies into other knowledge representation (KR) languages and can facilitate merging multiple ontologies [43]. The algorithm for OntoMorph is based on transformations. The transformations are a set of syntactic rewrites specified by pattern matching rules and semantic rewrites based on the KR models and inferable information. Each ontology is modified in such a way that the

combined morphed ontologies comprise a consistent merged ontology. Semantic rewrites require user input to interpret the models.

FCA-MERGE [44] is based on Formal Concept Analysis (FCA) [45]. The algorithm requires the extraction of instances from multiple documents that are written in terms of the ontologies to be merged. The extraction process returns a formal context that relates the relevant ontology to each document. With these contexts, FCA-MERGE creates a pruned concept lattice to represent the ontologies. The merged ontology is derived from the lattice with help from the user. FCA-MERGE does not consider the structural hierarchy of the source ontologies, relying mainly on lexical analysis of the documents.

The Ontology Alignment Evaluation Initiative (OAEI) sponsors annual Ontology Alignment Evaluation competitions [46], resulting in many promising ontology alignment algorithms, heuristics and interfaces. Lily was among the top performers in the 2008 contest. Lily [47] uses several techniques for determining mappings. Generic Ontology Matching is used for small ontologies; semantic subgraphs are created, then lexical and semantic similarities are evaluated and scored. Potential mappings with a score above a threshold, which can be influenced by a user-given weight, are accepted. Lily uses Large scale Ontology Matching for large ontologies, but its implementation is different and supposedly more time and space efficient for analyzing large ontologies than most other large scale ontology matching methods. Semantic Ontology Matching detects semantic similarities between ontologies by searching the World Wide Web for a set of potential mappings. These mappings are modified by Lily rules and a subset is selected as the alignment mappings. Lastly, Lily allows the user to debug and evaluate the returned mappings.

6.3 Previous Work at VIStology

Since its incorporation in 1997, VIStology has been involved in developing versatile software systems capable of restructuring their processing according to changing situations. This kind of capability is necessary for Level 2 information fusion in general [48,49,58] and for situation awareness in particular [50,51,52]. The versatility is achieved through a framework in which various tools and techniques are integrated. In particular, we use UML [53] to represent architectures of systems, their goals and domain knowledge; heuristic (AI) approaches to synthesize systems; formal methods to specify goals, systems and domain knowledge as well as analyze candidate solutions and systems; OWL, RDF and XML to capture and communicate knowledge; agent-based systems to achieve adaptability and mobility.

VIStology was involved in a DARPA-funded project to develop consistency-checking tools for OWL ontologies. This project, being done under a subcontract to Lockheed Martin for the DARPA DAML program has produced a system called ConsVISor [54] that helps with the checking of the consistency of ontologies represented in OWL. We are in the process of extending it with additional intelligence for identifying the root causes of inconsistencies and general errors.

In an earlier project for AFRL, VIStology applied formal methods to the problem of intrusion detection. This project resulted in a framework for an ontology and domain theory knowledge for multimode intrusion detection and fusion [55]. For the US Army CECOM, VIStology worked on a situation awareness system that included predicting a unit's geo position, displaying it on a map and minimizing the communication bandwidth (subcontract to Nova Research Corporation).

Both within and outside of VIStology, the authors of this report have an extensive research track record in the areas relevant to the proposal: formal reasoning about fusion, formal methods in software engineering, reasoning about uncertainty of decisions in fusion systems, architectures of goal-driven closed-loop adaptive systems, and in various applications of fusion. We have developed a formal approach to information fusion and published our results in journals and conference proceedings, cf. [56,57]. We investigated reference models for information fusion [58]. In [59] we dealt with checking consistency of systems specified in the UML language. In [60] we showed how to check consistency of the RM-ODP reference model. Our formal approach to the treatment of uncertainty in fusion systems has been addressed in [61]. We have conducted research in the area of intelligent agents that can reason about their goals and actions in [62]. In [63] we defined an extension to the UML meta-model, which would allow for the translation of UML formalized information to the representation in DAML (DARPA Agent Markup Language). We also investigated the use of ontologies for automatic target recognition [64,65]. We investigated the wavelet-based multi-resolution representations within the formal method approach and category theory in [57,66]. We investigated an approach to learning relations in [67].

7 Formalization of the Problem of Behavioral Situations

In this section we present our approach to formalize and model behavioral situations. First, we give a brief overview of the Espec category [70]. Then we relate this formalization to situation awareness. This is followed by a summary of our approach. And finally, we overview how we implemented behavioral situation modeling in Specware.

7.1 Evolving Specifications (Especs)

We recall that a specification is a set of *sorts* (or *types*) and *ops* (or *operations*), together called the *signature* of a specification, and *axioms*. The specifications form a category, Spec. The morphisms in Spec are called *interpretations*. The morphisms send sorts to sorts of corresponding types, ops to ops, and axioms to axioms and theorems. Note that in the category of specifications a given specification and its definitional extension (i.e., a bigger specification where we add new sorts defined in terms of old sorts) are thought of as isomorphic.

The category Spec is closed under colimits. The specifications and their colimits were used to produce functional programs that are provably correct [68]. Kestrel Institute developed the programming tool Specware that partially automated this procedure [69].

Our point of view is different, i.e., our goal is not to produce software but to model situation awareness scenarios. In dealing with situation awareness types of problems, we deal with a changing environment. The drawback of specifications is the lack of dynamism. One can have functions, and interpretations carry functions to functions, but underlying variables are not part of the morphism (only the corresponding sorts are). This feature can be improved by considering the category Espec of evolving specifications introduced in [70].

We recall the main construction from [70]. First it is beneficial to introduce guarded specifications.

Definition 1. The category Gspec of *guarded specifications* has the same objects as Spec but the morphisms $f : B \to C$ are defined by the *guarded interpretations*, i.e., the diagrams of interpretations

$$B \to B \wedge \Phi \leftarrow C$$

where Φ is the guard, i.e., the set of logical statements and $B \wedge \Phi$ is a specification B with the extra axioms given by Φ. Intuitively it means that the morphism f is defined only when the conditions Φ are satisfied.

We will represent the guarded interpretations graphically by the diagrams:

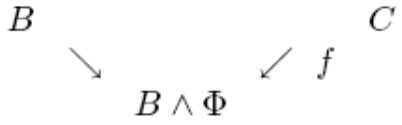


Figure 12. Guarded transition

The guarded interpretations are composed as follows. The composition of the above morphism with the following morphism

is the following diagram:

To introduce the category Espec we start with the *slice category* A/Gspec of guarded interpretations f : A → K, where A is a fixed specification and K is any specification in Spec. Next we introduce the category of *shapes*.

Definition 2. A *shape* is a directed graph G = (Vert(G), Edge(G)), where Vert(G) are the vertices and Edge(G) are the arrows of G, with the following properties:

a) It is reflexive, i.e $\forall a \in Vert(G)\ \exists i_a : a \to a$;
b) There is a distinguished initial node $x_0 \in Vert(G)$;
c) There is a collection of final nodes in Vert(G).

To a shape G we can associate a small category C_G.

Definition 3. An evolving specification F of shape G is a functor $F : C_G \to A/Gspec$ into the slice category of the guarded specifications.

The evolving specifications form the category Espec. The morphisms in this category are called refinements. Their description is technical and depends on another presentation of Espec as a comma category. Details of this category are given in [70]. The category Espec exhibits some nice properties, e.g., it is closed under colimits.

7.2 Application of Especs to Situation Awareness

In our application we will deal with situation awareness scenarios. We take the point of view of [71]. The goal of the system is to decide whether the incoming observations are consistent with a given type of situation and to give an estimate of likelihood that such situation is occurring.

The goal of this project was to develop the category theory based approach to tracking situations. The features that were important to us were specified in the following five requirements:

1. The developed framework should provide an ability to answer queries about the evolving situations.
2. Incorporating the dynamic aspect: the representation should capture evolving situations.
3. The dynamic aspect cannot be restricted to the monotonic logics.
4. Situation type classification should be part of the solution.
5. “Understanding” the situations, i.e. ability of inferring facts that are not explicit in the data, should also be covered.

The main difficulty one has to deal with is as follows. If one describes a situation as some kind of a theory involving logic and axioms, the morphism of two situations is not easy to define, as the change in situation can lead to axioms contradicting the original ones. However, checking that a given function defines a morphism of specifications is necessary, even though it involves a lot of theorem proving. Indeed, in order for the morphism to be well defined one needs to prove that axioms go to the theorems, i.e., one needs to prove these theorems in a corresponding theory. If this does not hold, the whole

specification is inconsistent and thus one can prove anything from such a specification. Thus inconsistencies must be avoided.

One of the key decisions that we had to make about the modeling of behavioral situations was what to treat as states. On one hand, we could try to model snapshots of the world as states. This, obviously, would not be tractable, since it would require including all of the knowledge about the whole world in each of the snapshots. Another possibility would be to use situations as states. But this would require repeating the knowledge associated with each situation as it evolves. Most of the knowledge about an evolving situation remains the same, yet some parts of it do change due to events received from the environment. The next level of simplification can be achieved by abstraction, i.e., by introducing an equivalence relation on situations that defines "situation types". Then states can be just situation types. In the following we first introduce the mathematical formulation of such an abstraction. Then this abstraction is utilized in the examples discussed in Section 8.

Definition 4. A Situation Type Classification is an Espec $F : C_G \to A/G$spec together with an equivalence relations $\sim=$ on vertices of G and on the arrows of G, satisfying the following compatibility relations:

a. If $x \in$ Vert(G) and $x\sim=y$, then $y \in$ Vert(G),
b. If $e \in$ Edge(G) and $e\sim=e'$ then $e' \in$ Edge(G),
c. If $e : x \to y$, $e' : x' \to y'$ are edges in G, and $e\sim=e'$ then $x\sim=x'$ and $y\sim=y'$. We denote $G_0 := G/\sim=$ to be the graph of equivalence classes (of vertices and edges) of the graph G.

The specifications F(x) ($x \in$ Vert(G)) are called *situations*. The guarded specification morphisms corresponding to edges $F(e) : F(x) \to F(y)$ are the *transitions*. The equivalence classes of situations F(x) are called the *situation types*. The equivalence classes of the transitions F(e) are called the *transition types*.

Notice that Definition 4 consists of two equivalence relations: on the vertices and on the arrows of G. This construction represents a state machine where the states are situation types and transitions are guarded interpretations.

Constructing situation types from an Espec is one of the principal goals of our theory. However for applications it is important to find an Espec that is manageable and can be implemented using our computer tools. Towards this aim, we can represent Especs of a group of "similar" situation types by one parameterized Espec which will capture some fixed knowledge of a given class of situation types, yet allowing for the specialization of the types by selecting specific values of the parameters.

In our approach we use ontologies to represent the knowledge of particular situation types. An ontology is a set of concepts - sorts (types or classes) and relations among the classes. Using ontological descriptions of the concepts and of the information about the environment (events) formalized in terms of the concepts of an ontology, one can derive conclusions on whether a particular situation type holds or not in a given world.

Roughly speaking, in the language relevant to situation type scenarios, we assume that the specifications F(x) have a set of finitely many relevant relations R1,…,Rs between their sorts, and the relation ~= is an equivalence relation on the possible s-tuples (R1,…,Rs). We use a specific ontology STO developed in this project (see [72]). The equivalence relation is defined in that language by an *elementary infon* [72]. So in our specifications we can use only the sorts, relations and rules that are *relevant*, as defined by the elementary infons. Thus the static part of the relation ~= is expressed entirely in the language of the STO ontology.

Definition 5. Let $F : C_G \rightarrow$ AGspec be an Espec. Let ~= be a Situation Type Classification with the quotient graph G_0. A Behavioral Situation Model (BSM) is an Espec $F_0 : C_{G0} \rightarrow$ AGspec of shape G_0 such that for every vertex $x \in Vert(G_0)$ the specification $F_0(x)$ has all sorts and axioms expressed in the language of the STO ontology.

7.2.1 State Machines of Situation Types and the Partial Implementation of Especs in Specware

The Behavioral Situation Model satisfies some of the five requirements given at the beginning of Section 7.2, while some others still require work. Referring to requirement 1, the language of ontologies is suitable for answering queries. Inference of implicit facts then is achieved by the use of VIStology's consistency checker (ConsVISor) and OWL inference engine (BaseVISor). This approach is described in Section 8. Specs are also implemented in the program Specware (discussed in this subsection), which includes a theorem prover (SNARK) that supports inferring facts that are not explicit in the representation. This approach is described in Section 8. However, this is still not done for a full implementation of Especs (thus requirement 2 is not fully satisfied). In our case, we have implemented the ontologies in OWL. Implementing the morphisms of ontologies understood as Specs is the subject of current work. Regarding the requirement 3, the nature of guarded specification morphisms allows for non-monotonic rewriting. The requirement 4 is expressed by our equivalence relation ~=. Finally, requirement 5, again is feasible thanks to the theorem prover included in Specware.

The main idea of our overall approach to behavior modeling is captured in the diagram of Figure 13. This figure is patterned upon the diagram of a guarded transition of Figure 12. ST_1 and ST_2 in this figure are two states in the state machine representation of a behavioral situation. As described earlier, these are Situation Types. Note Situation Type is a class in the Situation Theory ontology (STO). STO is the top vertex in this diagram. The three vertices are connected by arrows. The arrows denoted as st_1 and st_2 are morphisms in the category Spec. This means that STO is included into all the states in the state machine. In other words, all the states share the same common ontology. The double (curved) arrow between ST_1 and ST_2 represents a morphism in GSpec. This arrow represents a "guarded transition". This transition is defined by the two (straight) arrows between ST_1 and $ST_1 \wedge \Phi$, and between ST_2 and $ST_1 \wedge \Phi$. The first arrow is simply an import of ST_1 into $ST_1 \wedge \Phi$, while the f-arrow is the "rewrite" function which modifies

the axioms (the statements about the current situation) after receiving input from the environment.

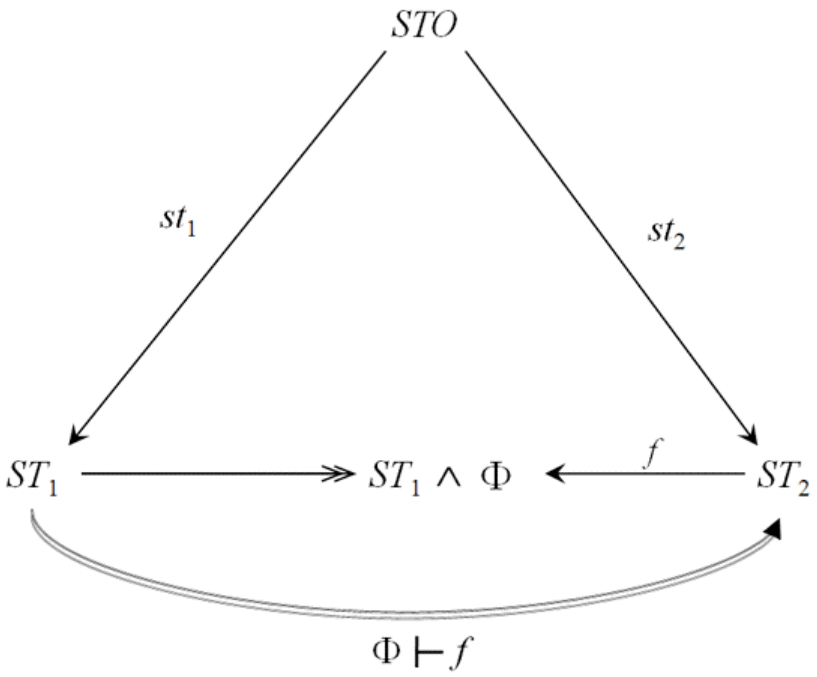


Figure 13. The main schema of Especs applied to Behavioral Situation Modeling

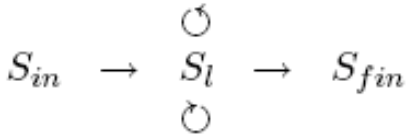
8 Behavioral Situation Modeling in Especs: Scenarios and Examples

In this section we present three examples of the application of our approach to the modeling of behavioral situations. The first scenario is taken from [70]. It deals with the problem of modeling the behavior of a computer program that computes the greatest common divisor (GCD). The second example deals with the modeling of a "Blocks World" in which the world changes in response to particular actions performed on the blocks in the world. The third example deals with a scenario that involves situations and with the tracking of a specific situation.

8.1 The Greatest Common Divisor (GCD) Example

Let us start with a simple Espec discussed in [70]. It is the program to calculate the greatest common divisor of two numbers. This state machine works by calculating the **gcd** of two input numbers by applying Euclidean algorithm. However we could use another Espec to reach the same goal.

The state transition graph G for this program is shown below



where S_{in} is the specification containing the numbers x_{in}, y_{in}, together with the axioms of the greatest common divisor z := gcd(x_{in},y_{in}). The state Sl contains four numbers x_{in}, y_{in}, x, y, together with the same axioms as S_{in}, and the axiom gcd(x,y) = gcd(x_{in},y_{in}). The arrow from S_{in} to S_l substitutes x := x_{in}, y := y_{in}. The arrow $S_l \rightarrow S_{out}$ occurs when x = y and it gives gcd(x,x) = x (which is one of the axioms of gcd). The two loops from S_l to itself act when x > y (resp. x < y), and substitute x – y for x (resp. y – x for y).

This section illustrates the use of Especs applied to a simple, well-defined problem by presenting the Epoxi state model for the GCD problem [70]. A state machine representation of the GCD problem from the Epoxi paper [70] is also shown in the figure below. The UML syntax is used in this figure (since a UML tool was used to draw it). So you can see the three *states*, as in the paper: One, Two and Loop. For each state, its spec (or *invariant*) is shown. For instance, for Loop, it is

gcd(X-in,Y-in)=gcd(X,Y)

Then you see transitions: initialize, Out, Loop1 and Loop2. For each transition, both a *guard* and an *effect* (the rewrite function f) are shown. For instance, Loop2 shows the guard [X<Y]. Guards are included in the brackets. The effect for Loop2 is /X:=Y-X. In other words, the effect (or action) is to assign the value of Y-X to variable Y.

In terms of Especs, states contain theories that hold in these states. So the term “invariant” is appropriate here. Guards are logical conditions that are denoted in the Epoxi paper as Φ. For us these are *events*, E. Since guards are logical conditions, we need to treat our events as axioms (logical sentences). Then the effects are the assignment functions, *f*, in the Epoxi paper. So this example (as well as the rest of the examples in the Epoxi paper) have a straight forward interpretation in terms of Especs.

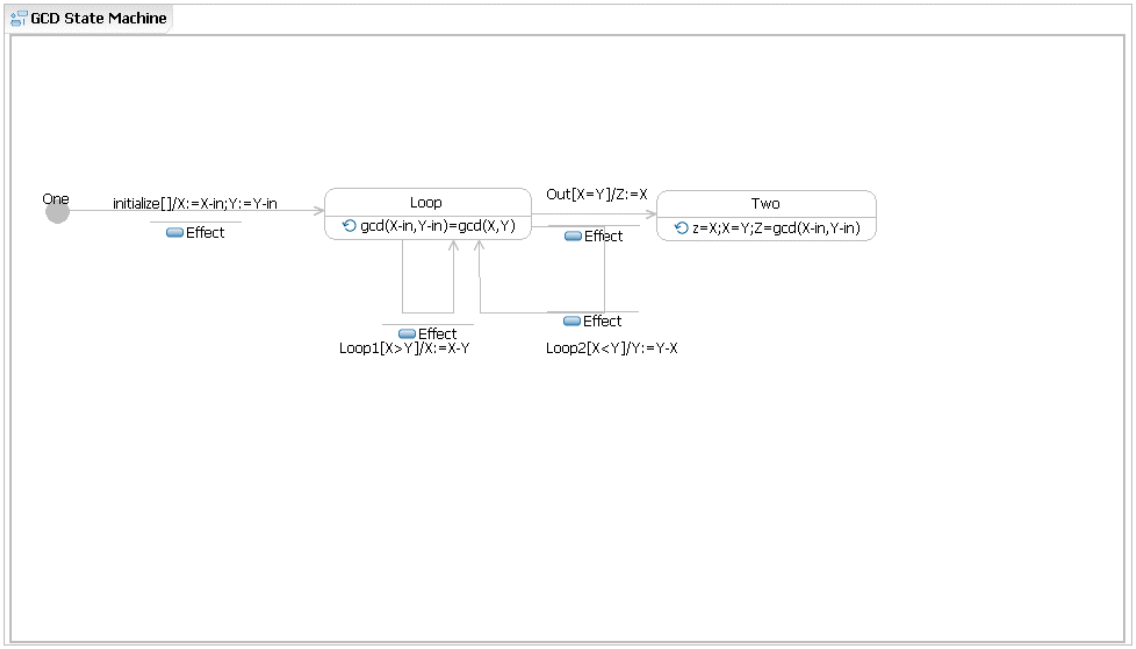


Figure 14 States and transitions of the GCD problem

8.1.1 Morphisms

So now the question is what are the morphisms in this Espec? It is assumed that a base spec (GCD-base) exists and is imported into the spec that specifies states. Since the spec for states does not introduce any new sorts, just variables and axioms, we would need to show that what holds in a given state (the invariant) should map to, and hold, in the invariant of the previous state + guard. Unfortunately, the Epoxi paper does not include details of the morphisms. So here are the full details of the morphisms.

Note that in the Epoxi approach, we are dealing with the category Espec in which nodes are of the same kind as nodes in the category Spec, but the morphisms are of a special kind. The Espec morphism between state One and Loop, denoted *initialize*, captures the transition between these two states. There is no guard for this transition, just an assignment X:= X-in and Y:=Y-in (this is the assignment function, *f*). Since there is no guard, the spec One ^ E is just One and the morphism i from One to One ^ E is just an identity. The morphism from Loop to One ^ E, again is just the morphism from Loop to One. This morphism can be represented by:

$$\{X \rightarrow X\text{-in}, X\text{-in} \rightarrow X\text{-in}, Y \rightarrow Y\text{-in}, Y\text{-in} \rightarrow Y\text{-in}, L \rightarrow O\}$$

where L is the axioms of Loop, i.e., the axiom: gcd(X-in,Y-in) = gcd(X,Y) and O is the axioms of One, i.e., the axioim: gcd(X-in,Y-in)=gcd(X-in,Y-in).

Now let us consider the transition between Loop and Loop, denoted as Loop1. This transition has a guard [X>Y]. The Loop1 morphism (in Espec) is represented by two morphisms in Spec:

$$\text{Loop} \rightarrow \text{Loop} \wedge [X>Y] \leftarrow \text{Loop}$$

The left morphism is just the trivial morphism. The right morphism is:

$$\{X \rightarrow X\text{-}Y,\ X\text{-}in \rightarrow X\text{-}in,\ Y \rightarrow Y,\ Y\text{-}in \rightarrow Y\text{-}in,\ L \rightarrow L'\}$$

where L is as before, but L' is a theorem that is obtained by rewriting L by the mappings of the sorts and variables:

$$gcd(X\text{-}in,Y\text{-}in)=gcd(X\text{-}Y,Y)$$

To insure that the morphism is correct, we just need to prove that L' holds. This can be proven in the following way. We know that in Loop the following holds (it is an axiom):

$$gcd(X\text{-}in,Y\text{-}in)=gcd(X,Y)$$

Now we need the following lemma.

Lemma:

$$gcd(X\text{-}Y,Y)=gcd(X,Y)$$

Proof of lemma:
gcd(A,B) is defined as the largest divisor of both A and B. Therefore, it is enough to show that the number D divides X and Y if and only if it divides X-Y and Y. So let D divide X and Y. This means that X=kD and Y=lD. Then X-Y=(k-l)D, so D divides X-Y and D divides Y. The other implication: if D divides X-Y and Y, then X-Y=sD and Y=lD, which implies that X=(s+l)D, so D divides X and D divides Y.

Loop2 is similar to Loop1 so there is no need to discuss this one. The last morphism (in Espec) is between Loop and Two, denoted by Out. In this case the guard is [X=Y]. The corresponding (right) morphism (in Spec) from Two to Loop is defined as:

$$\{X \rightarrow X,\ Y \rightarrow Y,\ Z \rightarrow X,\ \text{axioms(Two)}\text{—}\text{axiom(L)}\};\ \text{axiom}\ [X=Y]$$

The axioms of Two are: X=Y=Z and Z = gcd(X-in,Y-in). To prove that Two→Loop is actually a morphism, we need to prove that the axioms of Two become theorems of Loop. In other words, we need to show that, after the substitutions of X, Y and Z, as shown above, we have:

$$gcd(X\text{-}in,Y\text{-}in) = gcd(X,Y)$$

In other words, given X=Y and Z=gcd(X-in,Y-in), we need to prove:

$$gcd(X,Y)=gcd(X\text{-}in,Y\text{-}in).$$

Following the morphism mapping, we can rewrite this theorem as:

$$X=gcd(X,X).$$

It is obviously true.

In all of the examples in the Epoxi paper, they always import the base spec. In our case this would mean importing a whole ontology. The ontology would have to have some knowledge of all the objects in the particular states and thus we would not need to insert or remove individuals or classes to/from these ontologies. However, since the sets of axioms differ from one state to another, we still would have to deal with adding/removing facts to/from the state specifications, i.e., to/from ontologies associated with the particular states.

8.1.2 Events

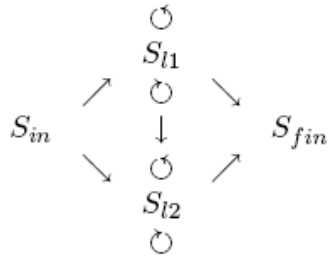
The Epoxi paper shows a number of examples of *events*. It is assumed that events are generated by an *environment*. So in each of the examples there is a spec that defines the environment. An environment spec is called a *parameter* and the specs that import such environments are called *parameterized specifications*. Environments are defined as procs (processes, or procedures).

Then a state machine (an Espec) listens to the events in some of the states. For instance, in the GCD example, the only state where communication with the external world takes place is in state One. The external world, in this example, provides the values of X-in and Y-in. The rest of the states do not listen to any external events. The final state (state Two) holds the final result Z.

In the Target Tracking example, the environment is defined in ReceiveCommands. It has just one state (state A). It continuously sends commands (!cmd), where cmd is a variable of type Event. The spec contains three states: Init, Receive and Dispatch. The Receive state listens to events and once it gets an event it transitions to Dispatch, where an action is taken, depending on the command received.

8.1.3 Equivalence Relations

We use the GCD example to demonstrate the notion of equivalence relations on Especs. To achieve this, instead of S_l we use two identical specifications S_{l1} and S_{l2} and have a map from S_{in} to S_{l1} if $x > 100$ and to S_{l2} otherwise. There would be two loops each from S_{l1} and S_{l2} to itself and one arrow from S_{l1} to S_{l2} substituting $x - y$ for x, when $x > 100$, $x - y \leq 100$. The Espec diagram for this state machine is shown below.



This diagram solves the same problem as the diagram shown in Figure 14 since the algorithm is the same. However, by introducing the relation $S_{l1} \sim = S_{l2}$ we see that the first

machine is more efficient, and the second can be reduced to the first. So our model could be used to improve the existing system.

8.2 Evolving World of Blocks

8.2.1 Problem Description

In this section we give an example of an Espec based on the example of state behavioral description of the Blocks world from [73]. This example is important for various reasons. First, we can implement it completely in Specware, including the theorem proving capabilities of SNARK [74], allowing us to check whether the transitions are defined (i.e. checking the correctness of the guards). The second reason for the importance of Blocks is that in this context we have natural examples of bisimulations, ideals and actions.

In the Blocks world there are three blocks - a, b and c - that are situated on top of the table. One block can be on top of another block, or can be directly on the table. We are not interested in the exact positions of the blocks, only in the relation telling us which block is on top of another block, which block is directly on the table and which block does not have anything on top of it (it is clear). Thus the states of the system correspond to different ways we can arrange our three blocks into piles.

This Espec has 13 states: 6 states where three blocks are on top of each other in one pile, 6 states where two blocks are on top of each other and one block is clear, and one state where all three blocks are directly on the table. We describe the state by three binary relations: the binary relation On and two unary relations Table and Clear. The relation On(a,c) will say that the block c is on top of a in the state in which it holds. The relation Table(a) means a is directly on the table in a given state. Finally the relation Clear (b) means that there is no block on top of b. In fact the relations Table, Clear are redundant, but having them simplifies rewriting procedures.

8.2.2 States, Events, Guards and Transitions in Metaslang

We will index the states by indicating by inequalities the boxes that are on top of each other. Thus we have six states where boxes are in one pile. These states are:
S(a<b<c), S(a<c<b), S(b<a<c), S(b<c<a), S(c<a<b), S(c<b<a).

There are six states with two piles. They are:
S(a<b;c), S(b<a;c), S(a<c;b), S(c<a;b), S(b<c;a), S(c<b;a).
Finally there is a state S(a;b;c) with three piles.

Our specifications will contain a common specification S, which can be described as follows. We use the Metaslang syntax (Metaslang is the language that Specware supports).

S has one sort (Block), three constants (a, b and c) represented in Metaslang as unary functions and three relations On, Table and Clear as shown below.

```
type Block
op a : Block
op b : Block
op c : Block
op On : Block * Block -> Boolean
op Table : Block -> Boolean
op Clear : Block -> Boolean
```

The axioms of S related to the relations Table, Clear and On are described below.

```
axiom not_y_on_x_if_x_on_y is fa(x:Block, y:{y:Block | y~= x} ) On(x,y) => ~(On(y,x))
axiom not_x_on_x is fa(x) ~(On(x,x))
axiom transitive_on is fa(x,y,z) On(x,y) && On(y,z) => On(x,z)
axiom not_x_on_table_if_x_on_y is fa(x,y) On(x,y) => ~(Table(x))
axiom not_y_is_clear_if_x_on_y is fa(x,y) On(x,y) => ~(Clear(y))
axiom diff1 is a ~= b
axiom diff2 is b ~= c
axiom diff3 is a ~= c
```

Additionally, S also contains three *events* that can be accepted by the states: M(a,b,c) – move block from a to c; U(a,b) – unstuck block a from b; S(a,b) – stack block a on top of block b.

The events and the state descriptions are related. We capture this relationship by the following axioms. The axioms are represented as event-condition-action rules. Below we show only the axioms that are related to the U event. As one can see, we write axioms for each of the predicates - On, Table and Clear.

```
axiom AxiomForU_Table is fa(x:Block, y:Block)
if U(x,y)                                                    %Event
then if On(x,y) & Clear(x)                                   % Condition
      then fa(z:Block) if z ~= x                             % Action
                       then Table'(z) = Table(z)
                       else Table'(z) = ~(Table(x)

axiom AxiomForU_Clear is fa(x:Block, y:Block)
if U(x,y)                                                    %Event
then if On(x,y) & Clear(x)                                   % Condition
        then fa(z:Block) if z ~= y                           % Action
                         then Clear'(z) = Clear(z)
                         else Clear'(z) = ~(Clear(z))

axiom AxiomForU_On is fa(x:Block, y:Block)
if U(x,y)                                                    % Event
then if On(x,y) & Clear(x)                                   % Condition
      then fa(v:Block, w:Block) if (v = x & w = y)           % Action
```

then On'(v,w) = false
else On'(v,w) = On(v,w)

Using the specification S we can specify the thirteen states. We give a description of two states – a state with three piles and a state with one pile of two blocks and one block on the table (clear). The descriptions of the rest of the states can be obtained by permuting a, b, c in the axioms.

```
S_a_:_b_:_c = spec                    % All 3 blocks on the table and clear
        import S
        axiom a is Table(a)
        axiom b is Table(b)
        axiom c is Table(c)
        axiom d is Clear(a)
        axiom e is Clear(b)
        axiom f is Clear(c)
        axiom g is fa(x,y) ~(On(x,y))
endspec

S_b_<_a_:_c = spec                    % One pile of two blocks and one block on table
        import S
        axiom a1 is On(a,b)
        axiom a2 is Table(b)
        axiom a3 is Table(c)
        axiom a4 is Clear(a)
        axiom a5 is Clear(c)
        axiom a6 is ~(Table(a))
        axiom a7 is ~(Clear(b))
        axiom a8 is fa(x:{x:Block | x ~= a}, y:{y:Block | y ~= b}) ~(On(x,y))
endspec
```

And finally we can describe transitions among the states. The transitions were defined in Section 7.2.1 and shown in Figure 13. Each transition a morphism in GSpec defined in terms of two morphisms in Spec and a node (a definitional extension), as shown in Figure 13. We repeat the diagram here with appropriate substitutions:

$$S\_b\_<\_a\_:\_c \rightarrow S\_b\_<\_a\_:\_c \wedge \Phi \leftarrow S\_a\_:\_b\_:\_c$$

The states S_b_<_a_:_c and S_a_:_b_:_c were defined above. So to complete this diagram we only need to represent the intermediary (definitional extension) node S_b_<_a_:_c $\wedge$ Φ and the two morphisms (arrows). The node for this transition is:

```
S_b_<_a_:_c_to_S_a_:_b_:_c = spec     % Transition event: U(a,b)
        import S_b_<_a_:_c
        import Event
endspec
```

The left arrow of this diagram was implemented via the direct “import” statement, i.e., the spec S_b_<_a_:_c was imported into the node’s spec. Another import statement brings Event into this spec (see discussion below). The right arrow of the diagram is implemented by an explicit “morphism” statement of Metaslang:

```
u_a_b = morphism S_a_:_b_:_c -> S_b_<_a_:_c_to_S_a_:_b_:_c
{
        On +-> On',
        Table +-> Table',
        Clear +-> Clear'
}
```

This requires some additional explanation. Recall that morphisms must map sorts and ops in such a way that axioms of the source spec become theorems in the target specs. Recall also that in our problem each transition means a rewrite of the axioms, i.e., the axioms of the source state are rewritten in the target spec by the rewrite function f. To avoid logical inconsistencies we redefine the ops of the target spec in the definitional extension spec. In this particular example, the ops On, Table and Clear are redefined as On’, Table’ and Clear’ in the definitional extension state S_b_<_a_:_c_to_S_a_:_b_:_c. The guard Φ captures two facts: (1) that an event took place (e.g., U(a,b) in this example) and (2) that the preconditions for this event are satisfied so an action can take place. These conditions were shown in spec S above, where the results of the events were represented as event-condition-action rules. The actions are the rewrites (function f). These rewrites were also defined in spec S (the “action” parts of the event-condition-action rules).

Events come from the environment. In our formalization they are captured in a spec called Event. This spec contains only one of the possible events at a time. So in the example below, only U(a,b) is active. The spec also shows two other potential events, but they are not active since they are commented out in this spec.

```
Event = spec
        import S
% Current event
        axiom UnstackAxiom is U(a,b)
%       axiom StackAxiom is S(a,b)
%       axiom MoveAxiom is M(c,b,a)
endspec
```

8.2.3 Inference Over the Espec Representations

The next step in our Metaslang implementation of the state machines is to show that the system “understands” the problem, i.e., to show that the system can infer some facts that are not explicitly represented in the formalism. We achieve this goal by using a theorem prover (SNARK), which is integrated with Specware.

To use inference, we need to specify our queries. In Metaslang they are represented as conjectures. For instance, we could add the following conjecture to the specification of state S_a_:_b_:_c:

```
conjecture b_is_clear is
        Clear'(b)       % <= axiom e in S_a_:_b_:_c
```

This conjecture means that after the transition from state S_b_<_a_:_c to the new state S_a_:_b_:_c, block b is clear. To verify this we need to invoke SNARK. This is achieved by stating such a requirement in spec S_b_<_a_:_c_to_S_a_:_b_:_c, which formalizes this transition:

```
P1 = prove b_is_clear in S_b_<_a_:_c_to_S_a_:_b_:_c
```

The result of an invocation of SNARK on a specification like the one described above is shown below. This is a trace of the activity generated by Specware and SNARK. The first line shows a request to process the file called blocks6.sw. The last line shows that SNARK was successful in proving this conjecture.

```
Specware Shell
* proc /Users/Mitch/Vis/SiBMod/FinalReport/blocks6.sw
;;; Elaborating spec at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#Event
;;; Elaborating spec at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#Q
;;; Elaborating spec at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#Q_a_:_b_:_c
;;; Elaborating spec at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#Q_b_<_a_:_c
;;; Elaborating spec at
/Users/Mitch/Vis/SiBMod/FinalReport/blocks6#Q_b_<_a_:_c_to_Q_a_:_b_:_c
;;; Elaborating spec-morphism at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#u_a_b
;;; Elaborating proof-term at /Users/Mitch/Vis/SiBMod/FinalReport/blocks6#p2
   Expanded spec file: /Users/Mitch/Vis/SiBMod/FinalReport/Snark/blocks6/p2.sw
   Snark Log file: /Users/Mitch/Vis/SiBMod/FinalReport/Snark/blocks6/p2.log
p2: Conjecture b_is_clear in Q_b_<_a_:_c_to_Q_a_:_b_:_c is Proved! using Snark in 0.2
seconds.
*
```

This conjecture happens to be one of the axioms in the target spec. Since to be a morphism, it is necessary to prove that all the axioms of the target spec are theorems of the transition spec, one can achieve this in Specware by requesting “proof obligations”. This is shown in the capture below.

```
u1 = obligations u_a_b
p1 = prove a in u1
p2 = prove e in u1
p3 = prove g in u1
p4 = prove AxiomForU_Table in u1
p5 = prove AxiomForU_On in u1
p6 = prove AxiomForU_Clear in u1
```

p7 = prove not_y_on_x_if_x_on_y in u1 % fails
p8 = prove f in u1

This completes our presentation of the inference capability for the Blocks world scenario.

8.3 Bridge Explosion Scenario

8.3.1 Scenario Description

To explain the main ideas of our approach we present a relatively simple scenario. In this scenario an analyst poses a query related to the 2010 Boston Marathon: “Will there be a bridge explosion during the 2010 Boston Marathon?” The goal is then to support the analyst in not only answering such a query at the time the query is posed, but most importantly, to track the status of this query as new evidence is gathered. The objective of then is to indicate how such tracking can be implemented.

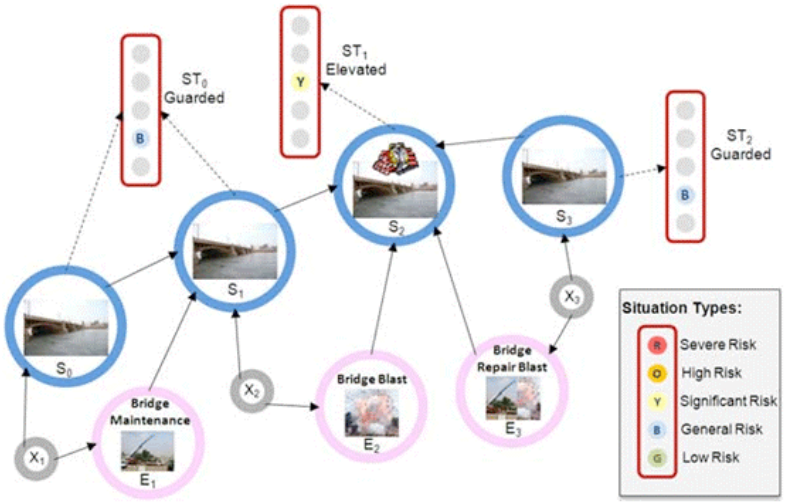


Figure 15 The Bridge Explosion Scenario

Once such a query is posed, the system supports the analyst with expressing the query in the language defined by the Behavioral Situation Theory Ontology (STO-B). This is followed by an interactive process of accepting new evidence (events), resolving ambiguities in the representation of the current situation and the incoming events, merging of new events with the current situation and automatic derivation of consequences of new evidence and particular representational decisions.

A graphical depiction of the scenario is shown in Figure 15. Only major phases of the whole process are shown in this figure. The left-most circle represents the situation that the system is aware of after the analyst posts the query. Since the analyst is already concerned about some danger of the potential situation, the lights above the circle indicate that the situation type can be classified as “General/Guarded Risk” (or code Blue). The circle labeled “Bridge Maintenance” represents the event that arrives at some later time. This event includes a relation (undergoes) between the bridge and the activity of Repair. The system needs to combine the current description of the situation (S0) with

this new event (E1). In order to resolve possible ambiguities, a mapping needs to be provided that indicates which of the terms in the description of the current situation and in the event are different names for the same concepts. The mapping is indicated by the arrows from a smaller circle to the two large circles. The meaning of the small circles is that these are sets of terms. The meaning of the arrows from the small circles is that a pair of arrows originating from the same terms in a small circle point to two terms in the large circles that need to be unified, i.e., must be treated as the same term. The merging creates a new description of the current situation, however, since this event describes bridge maintenance, the type of the situation does not change; it still remains at the same level of dangerousness. The second event, “Bridge Blast”, includes the isPlaceOf relation between Bridge and Blast, and is processed in the same way as the previous event. Since isPlaceOf relates bridge to an explosion, the level of danger (type of situation) is raised to “Significant/Elevated Risk ” (or code Yellow). Finally, the third event delivers a message that the blast was actually related to the repair. At this point, some information is actually subtracted from the current description of the situation. Note that the direction of the arrows in this case is different than in the previous two events. This time the arrows point towards the previous situation, indicating that the sum of the event and the new situation add up to the previous situation.

8.3.2 Behavioral Situation Development and Tracking Architecture and Process

Our models and tools utilize OWL (Ontology Web Language) based data structures, providing type reference for objects, relations, and situations. Figure 16 shows meta-level ontologies (left-hand side) and high level situation classes and relations (right-hand side). They provide both the reference and high-level templates for OWL-driven situation development and tracking. The arrows denote inheritance and/or derivation type associations. For example Utterance Ontology inherits some classes and relations from the left-hand side ontologies. However, its specific objects (instances) may come from another source (such as a plain text document).

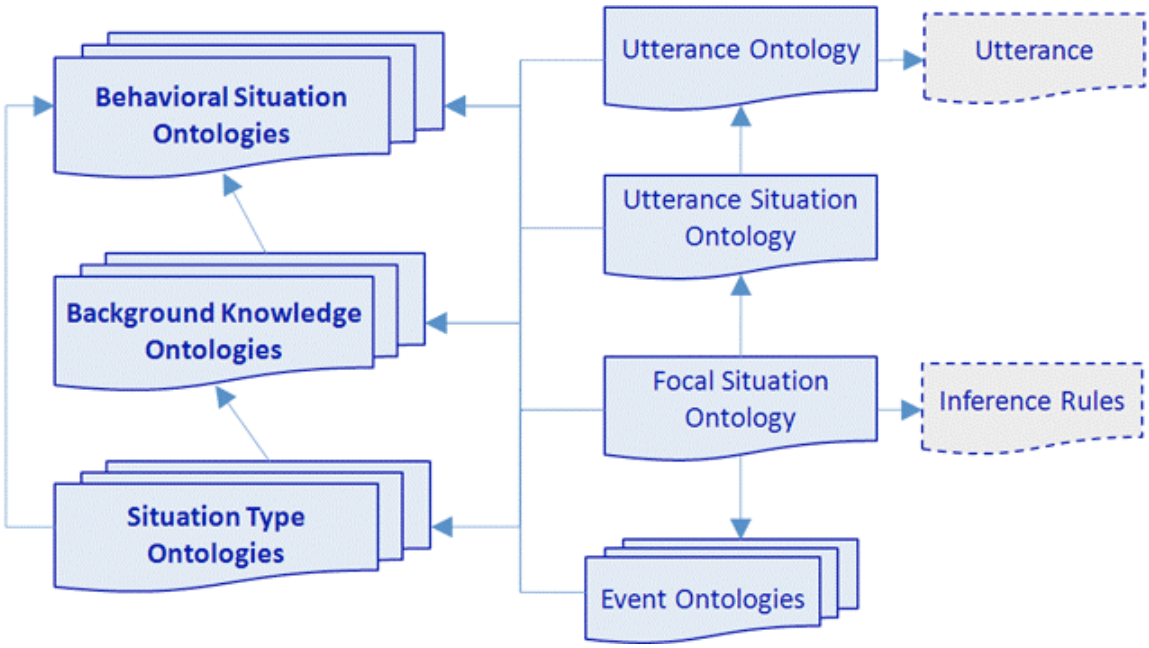


Figure 16 Reference Ontology Documents

Situation development and tracking requires careful crafting of various types of documents and procedures. Since our approach relies heavily on ontologies expressed in OWL it ensures that all entities we process within our system are formally defined. Nonetheless, we have faced numerous challenges in an attempt to formalize or automate the situation development process. At this point, the process is quite interactive and requires a substantial amount of human involvement and intervention. Figure 17 shows our general approach to the situation development process in which ontologically expressed data structures play a predominant role.

There are three types of "users" participating in the process: *Investigator*, *Ontologist*, and *Situation Analyst*. In a real-world application, there may be many more human-constituencies involved. Investigator may be represented by a leader or group of leaders interested in finding answers to their important questions (queries or goals). He or she may be an intelligence analyst who wants to assess possibilities or probabilities of some threats associated with public, industrial, social, political or warfare events. Ontologist is an expert in building OWL documents. His/her/their role is to make sure that all relevant data is captured formally in OWL. Our current system stores such data in Web-ready files, each containing one ontology. Multiple ontologies are combined (merged) using the OWL-import mechanism. Finally, Situation Analyst is a person or group of people, specializing in situation development and tracking. He/she/they have a good understating of the subject matter and situation formalization methods and tools.

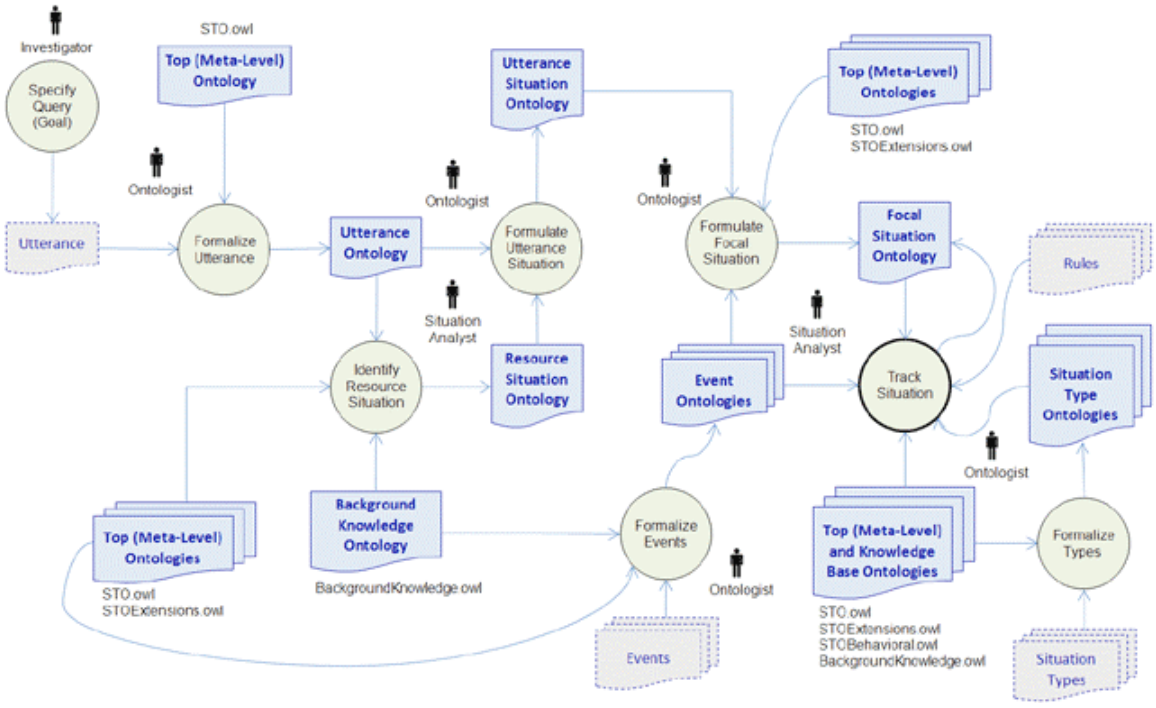


Figure 17 Ontology based situation development and tracking.

The entire process starts formally with an initial query captured more or less formally as an utterance (stored in an *Utterance* document). Using our *Situation Theory Ontology* (STO, http://vistology.com/ont/2008/STO/STO.owl) as a meta-reference, the ontologist expresses the original query as *Utterance Ontology*. At this stage, the utterance is still represented as a plain text (string) but specific classes, objects and properties are formally defined: sto:Utterance, sto:Agent, sto:Sentence, sto:Value, sto:Time, *sto:utteredBy*, *sto:hasAttributeValue*, *sto:attributeValue*. All the classes and properties are formally defined in STO.

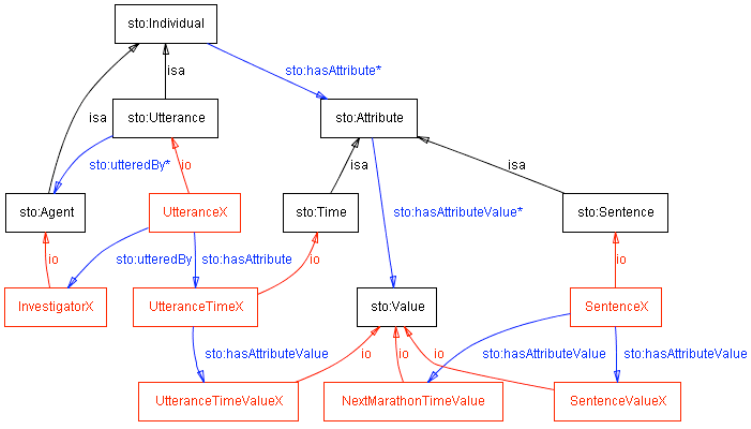


Figure 18 The Utterance Ontology

In the next phase, formal situations are introduced. Ontologist and Situation Analyst identify a Resource situation (Figure 19) and formulate an Utterance situation. Both the situations are expressed in OWL.

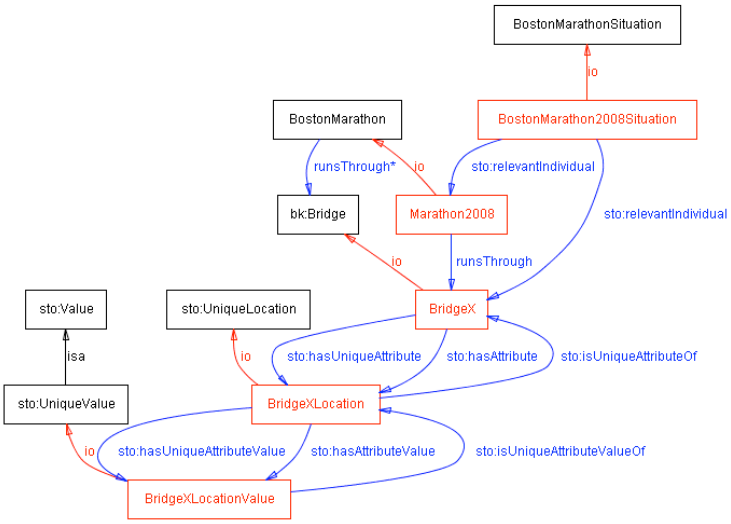


Figure 19 The Resource Situation Ontology

The Resource situation captures objects and relations that the utterance is referring to. The original query, expressed as the utterance ontology (Figure 18), directly refers to the next Boston Marathon which is defined by the Boston Marathon situation (Figure 19). This ontology captures formally important (focal) types (classes) and properties. In particular, as shown in Figure 19, class Bridge, having instance BridgeX at a specific (fixed/unique) location, are of our special interest.

Once the resource situation is defined, the utterance can then be expressed also as a situation, giving rise to Utterance Situation Ontology (Figure 20). This is a milestone step in that the Utterance situation creates a criterion of the relevance.

The Utterance ontology is a high-level formalization of the original query uttered by the agent. It serves as a basis (along with the Boston Marathon ontology) for development of a specific utterance situation. One can infer from the query objects such as: BridgeX, ExplosionX, BostonMarathonX, as well as predicates related to a blow-up, happening during the Boston Marathon. Thus the query can be expressed, using RDF statements (triples of subject, predicate, and object):

1. [ExplosionX] [blowsUp] [BridgeX].
2. [ExplosionX] [happensDuring] [BostonMarthonX]

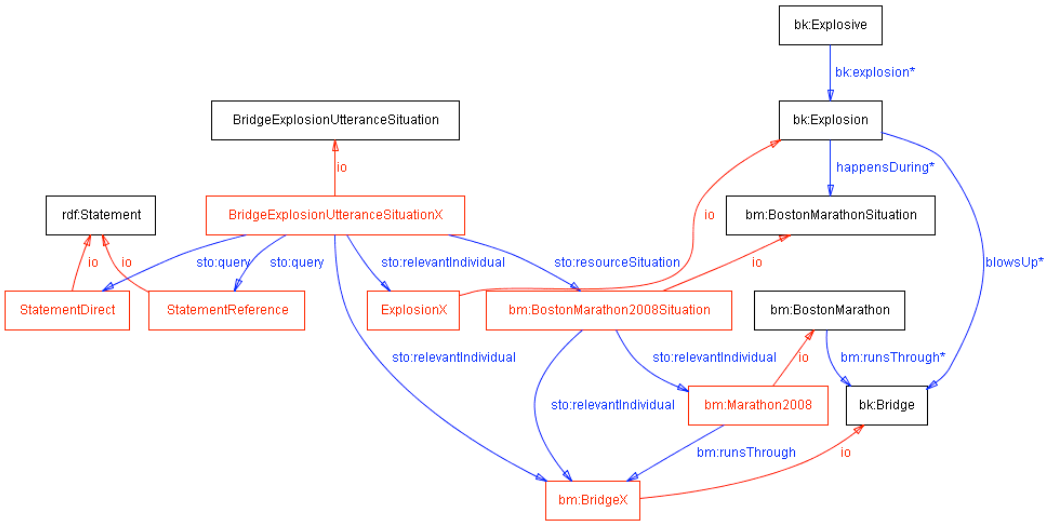


Figure 20 The [Bridge Explosion] Utterance Situation Ontology.

We can now move to the next steps where we will attempt to describe the fragment of reality the Utterance situation is about. Such a description is done in form of Focal Situation Ontology. Initially, the Focal situation does not have to bring any new information. It simply sets itself in the context of the Utterance situation. This whole process, so far, is consistent with the Situation Theory of Barwise [14]. However, it sets the process running through the model of reality expressed in OWL which is a powerful descriptive tool but, more importantly, it enables us to deal with the issue of relevance with assistance of automated or semi-automated reasoning procedures.

The Focal situation emerges initially from the Utterance situation. This situation is part of a dynamic system, in which it evolves by being augmented through Situation Analyst's background knowledge and new arriving events. Both the background knowledge and events are also expressed as ontologies. Moreover, each event is treated as a situation. Equipped with the high-level reference, represented by STO, STOBehavioral, STOExtension and BackgroundKnowledge ontologies, Situation Analyst can track the Focal situation which is being exposed to and eventually altered by new events (Event situations). Each new event may alter the Focal situation. As we put it, each event may bring the situations to another state (or situation type). The next state of the situation may be determined by the Situation Analyst or by a software agent. In any case, situation transitions are carried out based on more or less formal rules that take into account types (classes) objects (individuals) and relations (properties) of the Focal situation (in the current state) as well as of the new Event situation, along with those set in the Utterance situation.

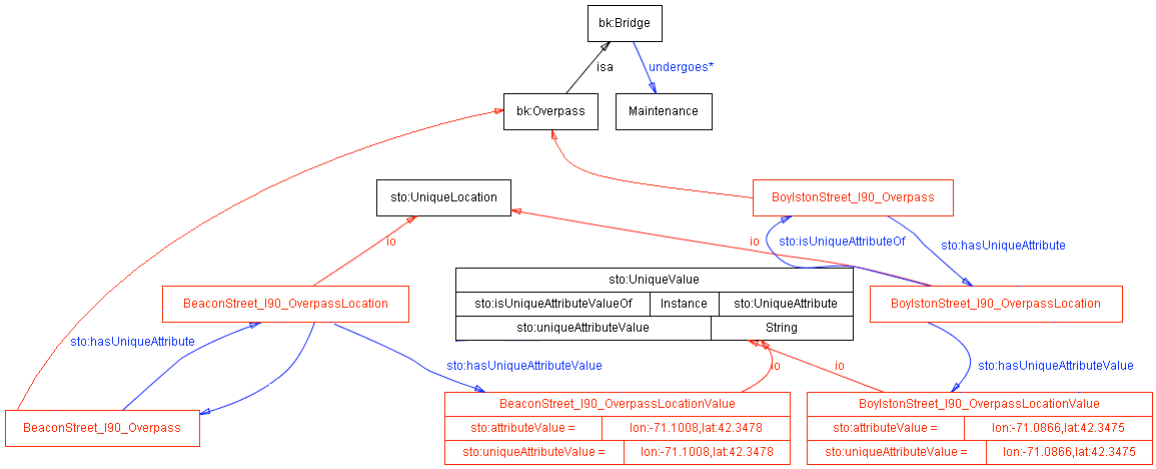


Figure 21 The Maintenance [Event] Situation Ontology.

The focal ontology combines relevant objects and relations (from the utterance situation) with events (event type situations) that come to the agent's attention from external information sources. Suppose that the agent focuses on a situation that involves some bridge maintenance activity in Boston. This event brings information about maintenance activities (repair and inspection) of two overpasses in Boston (Figure 21).

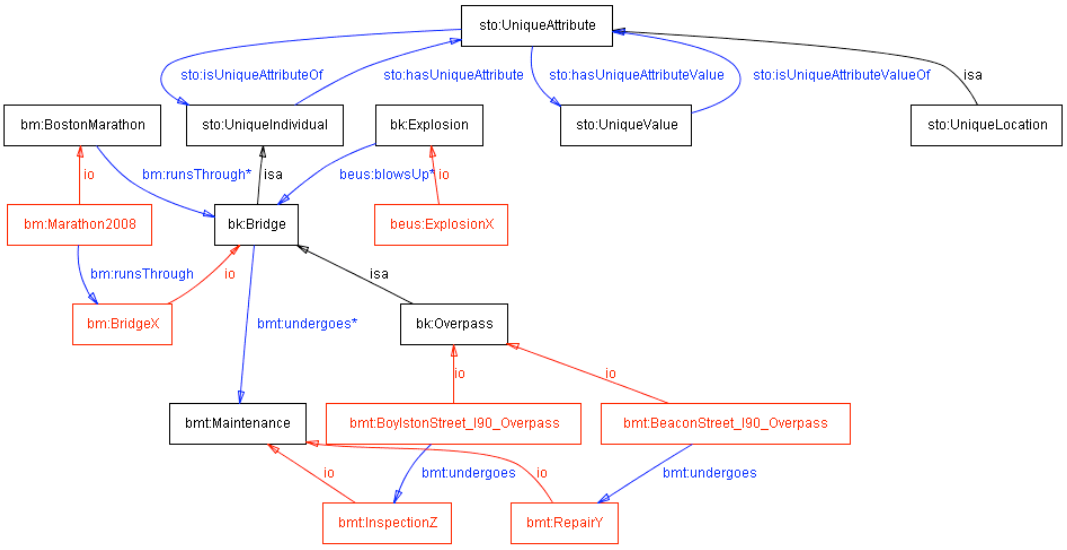


Figure 22 The Focal Situation Ontology.

It is interesting to note that one of the overpasses in the Bridge Maintenance situation has the same location value as the bridge mentioned in the Boston Marathon situation. Are they really the same individuals? In order to answer this question, we add assertions to the Focal situation ontology (Figure 22), using the OWL-based *sameAs* property.

We implement focal situations in ontologies by *importing* the Utterance situation and one or more event-type situations (notice that the Resource situation is already included in the Utterance situation). Entity types that are common to any situation are defined in the STO. Entity types specific to the "Bridge Blowup" scenario are defined in our Background Knowledge Ontology, (BKO). A fragment of the BKO is shown in Figure

23. Most of the entities that appear in our utterance, focal and event-type situations are based on the entities of the two ontologies (STO and BKO). The BKO is supposed to capture the agent's understanding to the object types and relations/properties involved in the situations. Some external (event-type) situations, of which the agent will become aware, may define additional (new) types to be reconciled (reason about) when merged with the STO and BKO and the Utterance situation.

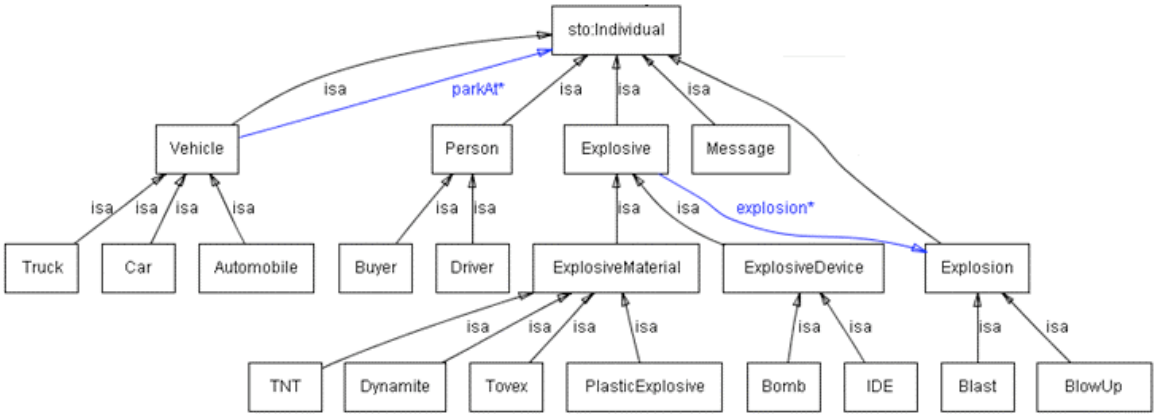


Figure 23 The Background Knowledge Ontology

Notice that in the BKO, class Overpass is a subclass of class Bridge. Since the locations of the bridges (overpasses) are defined by OWL object and data-type properties (as shown in the STO), our assertions must be specified for each instance that is used to specify the locations. Consequently, we add the following statements to the Focal ontology:

```
<rdf:Description rdf:about="&bm;#BridgeX">
   <owl:sameAs rdf:resource="&bmt;#BeaconStreet_I90_Overpass"/>
 </rdf:Description>
 <rdf:Description rdf:about="&bm#BridgeXLocation">
   <owl:sameAs rdf:resource="&bmt;#BeaconStreet_I90_OverpassLocation"/>
 </rdf:Description>
 <rdf:Description rdf:about="&bm;#BridgeXLocationValue">
   <owl:sameAs rdf:resource="&bmt#BeaconStreet_I90_OverpassLocationValue"/>
 </rdf:Description>
```

Next, we use our consistency checking program, ConsVISor [54], to test these assertions. It turns out that the assumption of the two individuals being the same is consistent. Interestingly, a similar test for a *differentFrom* relation results in inconsistency.

Checking (querying) the Focal situation ontologies for consistency should be considered as an initial step in the agent's situation analysis. Depending on the outcomes of the consistency checking, some event-situations may be eliminated as not feasible.

Reasoning about situations involves identification of common or different classes (types), individuals and properties (relations). The advantages of expressing the situations in OWL/RDF/RDFS lie in OWL's natural reasoning capabilities [75]. The following are just a few examples for such capabilities:

- Implication based on inheritance:
  ```
  (rdf:type I-91BeaconOverpass Overpass)
  (rdfs:subClassOf Overpass Bridge)
  ```
  imply
  ```
  (rdf:type I-90BeaconOverpass Bridge)
  ```
- Inference based on the `rdfs:range and rdfs:domain` constraints:
  ```
  (rdfs:domain isPlaceOf Bridge)
  (rdfs:range isPlaceOf Explosion)
  (isPlaceOf I-90BeaconOverpass BostonMarathon2010Explosion)
  ```
  imply
  ```
  (rdf:type I-91BeaconOverpass Bridge)
  (rdf:type BostonMarathon2010Blowup Explosion)
  ```
- Implication based on the functional property:
  ```
  (rdf:type uniqueAttributeValue owl:FunctionalProperty)
  (uniqueAttributeValue Bridge 90BeaconOverpass)
  (uniqueAttributeValue Bridge 90BeaconBridge)
  ```
  imply
  ```
  (owl:sameAs 90BeaconOverpass 90BeaconBridge)
  ```

In short, the entire, ontology-based, situation transition process is shown in Figure 24.

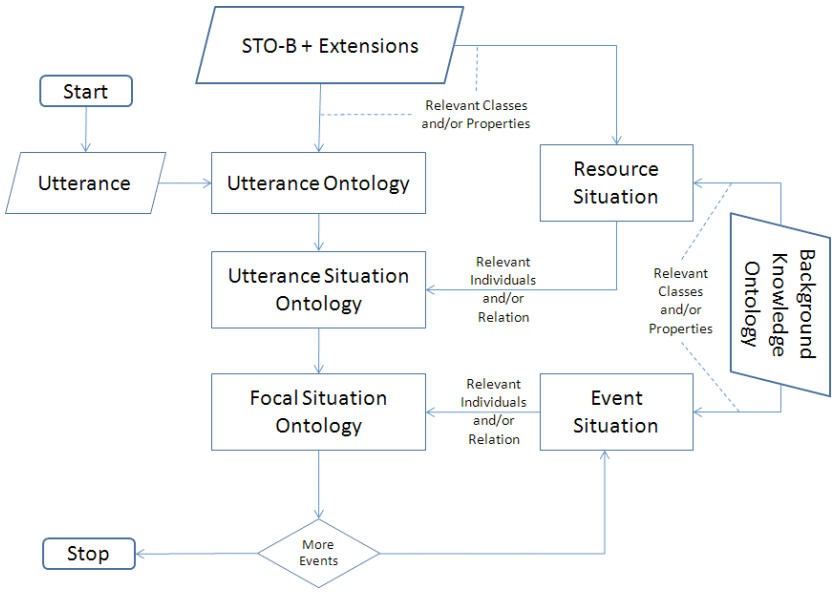


Figure 24 The ontology-based situation transition process.

The next section shows important aspects of processing the focal ontology, transitioning from one situation type to another.

8.3.3 State Machine Based Example

In our approach to situation tracking, change−notification events arrive at the situation−tracking agent, one at a time. If the event satisfies certain criteria, it is then merged with the current situation, resulting in a modification of the situation, including possible reassignment to a different situation type.

The process is described by introducing the proof-of-concept scenario. The scenario involves three states related to a bridge: NormalBridgeOperation, ClosedForMaintenance and BridgeUnderThreat. Each state is modeled as a situation type, capturing the objects and their relations to one another for every situation in that state. The states have one common object, the bridge whose behavior is of primary interest.

Events can trigger transitions from state to state. Figure 25 depicts the situational states and the four possible state-to-state transitions, shown as arrows from the beginning state to the ending state. Each possible transition is associated with an event category unique to the initial state and final state pairs. If an incoming event corresponds to one of the categories, a situation may transition to another state.

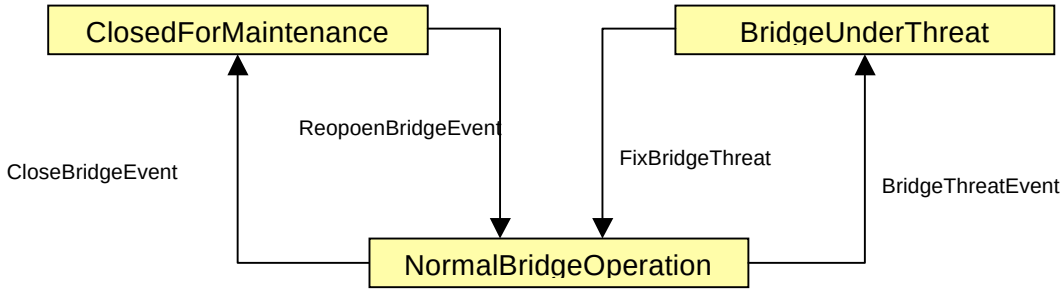


Figure 25 Situation states and transitions between situations for a bridge.

In this scenario, NormalBridgeOperation represents the state when the bridge is open and functioning as usual. When maintenance is performed on the bridge, the situation changes to ClosedForMaintenance, which signifies an ongoing maintenance. Completion of the maintenance allows the bridge to reopen and resume its normal operation. While in NormalBridgeOperation, if an explosion occurs on the bridge, the situation transitions to BridgeUnderThreat. Once the bridge is fixed, the bridge reopens and returns to operating normally.

Realistically speaking, it is possible for an explosion to blow up the bridge while it is closed for maintenance. It may also be possible that multiple maintenance projects are in process at the same time. In this case, even if one maintenance operation is finished, the bridge does not yet reopen. This results in a self-loop transition from ClosedForMaintenance to ClosedForMaintenance, with the completed maintenance project removed from a list of ongoing maintenance efforts. The situation cannot transition to NormalBridgeOperation until every maintenance operation is done. Likewise, there may be several explosions on the bridge, each requiring a separate repair. Each repair job causes a transition back to BridgeUnderThreat, until all the explosions are addressed and fixed. These cases are omitted for the scenario since the processing of these transitions is similar to the implemented transitions.

The scenario is embedded in the formalism of State Machines. The specifications of the situational states and event categories are designed according the principles of Especs. The spec for NormalBridgeOperation is shown in Figure 26. The specs were created using SpecWare, which is useful for checking syntax. All classes, or sorts, are declared in

a common spec, so the morphisms only require the mapping of instances to instances and axioms in the new state to theorems in the old state.

```
S0 = spec
      import Base
      import Background

      op NormalBridgeOperation : SIT
      op Bridge0 : Bridge
      op Bridge0Location : UniqueLocation
      op Bridge0LocationValue : UniqueValue
      op Bridge0LocVal : string
      op S0_EInfon : ElementaryInfon

      axiom    Bridge0IsRelevantIndividualOfNormalBridgeOperation    is
relevantIndividual(NormalBridgeOperation, Bridge0)
      axiom       openIsRelevantRelationOfNormalBridgeOperation       is
relevantRelation(NormalBridgeOperation, open)
      axiom openIsImpliedByS0Rule is impliedByRule(open, S0Rule)
      axiom Bridge0IsOpen is open(Bridge0)
      axiom         Bridge0HasUniqueAttributeBridge0Location         is
hasUniqueAttribute(Bridge0, Bridge0Location)
      axiom          Bridge0LocationIsUniqueAttributeBridge0          is
isUniqueAttributeOf(Bridge0Location, Bridge0)
      axiom           Bridge0LocationHasUniqueLocationValue           is
hasUniqueAttributeValue(Bridge0Location, Bridge0LocationValue)
      axiom  Bridge0LocationValueIsUniqueAttributeValueOfBridge0Location
is isUniqueAttributeValueOf(Bridge0LocationValue, Bridge0Location)
      axiom  Bridge0LocationValueHasUniqueAttributeValueBridge0LocVal  is
uniqueAttributeValue(Bridge0LocationValue, Bridge0LocVal)

      axiom          S0_EInfonSupportsNormalBridgeOperation          is
supportedInfon(NormalBridgeOperation, S0_EInfon)
      axiom  openIsRelevantRelationOfS0_EInfon  is  relation(S0_EInfon,
open)
      axiom S0_EInfonHasAnchorBridge0 is anchor1(S0_EInfon, Bridge0)
      axiom S0_EInfonIsTrue is polarity(S0_EInfon, _1)

endspec
```

Figure 26 Spec for the state NormalBridgeOperation

8.3.3.1 Ontologies

This section details the implementation of the concrete scenario and the situation tracking prototype tested on the scenario.

Each state, event category, situation and incoming event is captured by an ontology and expressed in terms of STO and a modified version of STO-B. The Extended Situation Theory Ontology (STOExtension) includes several classes and properties to support the inference that instances which are named differently or exist in separate namespaces refer to the same objects. The ontologies for the concrete scenario are available in OWL format at http://vistology.com/ont/2009/especs/.

BackgroundKnowledge.owl contains domain knowledge specific to the scenario, e.g., the class Bridge and the relation open. This is equivalent to the common specification SpecA in Especs [70]. BackgroundKnowledge is represented in STO and STOExtension. While STO includes various situation types, the examples use three types: Situation, FocalSituation and Event. As described in Figure 25, the states: NormalBridgeOperation, ClosedForMaintenance and BridgeUnderThreat are situation types, i.e. they are subclasses of Situation. Similarly, the event categories are types of Events. The focal situation in a particular state is an instance of the situation type. In the same way, an incoming event meeting the necessary and sufficient conditions of the event category is an instance of that event type.

Each situation state serves as a 'template' that is applicable for any situation satisfying the requirements of the state. In the bridge scenario, every state has one common object – the bridge. The initial focal situation introduces the specific bridge whose behavior is tracked. As is common in ontology authoring, instances in different ontologies may be named differently but refer to the same object. To indicate that incoming events involve the same bridge in the focal situation, the bridge instances must have the same unique location value. Likewise, maintenance instances also have a unique ID to verify that the completed maintenance indicated in an event is the same maintenance that caused the bridge closure.

Figure 27 shows a graphical depiction of the state NormalBridgeOperation. A focal situation is initially in this state, with a unique location longitude / latitude value to uniquely identify the bridge of interest. When an event arrives, one of the conditions considered is whether it is related to the same bridge, i.e. the bridge has the same location value. If not, the event is ignored as irrelevant to the situation.

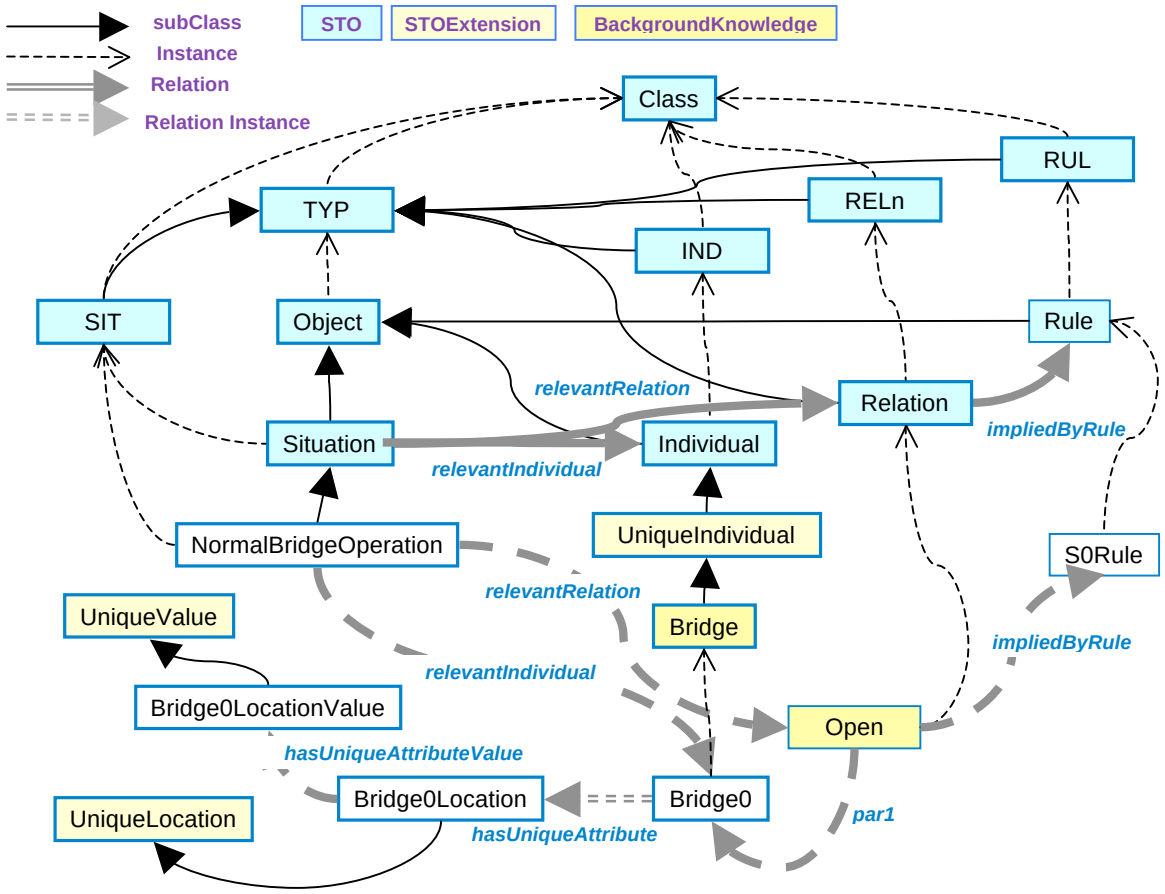


Figure 27 State for the normal operation of a bridge

The ontology capturing the situation state also includes all necessary information about the relevant individuals and relations. Each situation and event type has at least one relevant relation that is essential to describing the situation. This information, represented as RDF triples, comprise the “axioms” of each state, i.e. the set of triples that must exist in the situation’s ontology for that situation to be in the state. When trying to determine the state of a particular situation, these “axioms” can be considered as the conditions that are necessary and sufficient for a situation to be in a state. These conditions are defined in a rule associated with the relevant relation. The event type of an incoming event is likewise specified in a rule that implies the relevant relation of the event category. The rules are written as strings in the OWL ontology.

The body of the rule contains the requirements that must be true for an instance to be of that situation or event type. For states, the head of the rule simply asserts that the instance is in the state. For events, the head defines the rewrite function and the morphism. The purpose of the rewrite function is to modify the focal situation in order to transition to the new state. This may require incorporating information from the incoming event, i.e. merging the two ontologies in such a way that maintains consistency and does not violate the axioms of the new state. The rewrite may also require changing parameters of the

situation or removing information from the situation. For example, in transitioning from ClosedForMaintenance to NormalBridgeOperation, the information about the now-completed maintenance is no longer needed in the focal situation. The second part of the rule head describes the morphism, i.e. the mappings of instances to instances and the axioms of the new state. The matching theorems can be determined by applying the mappings to the axioms.

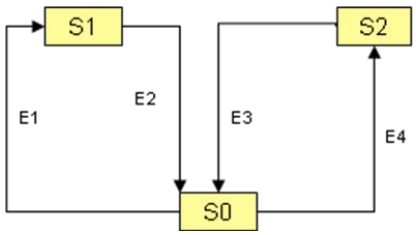


Figure 28 Relations between situation types and events.

An example of this construction is the Bridge BSM with three situation types S0, S1, S2 and four types of events E1, E2, E3, E4 shown in Figure 28.
A BSM type model can be used to run simulations which can lead to prediction of the behavior of the actual system. We expect to develop the algorithms to refine the BSM's using those simulations. For example if two situations presumed to be equivalent, give different types after a series of the same events, this means their types have to be distinguished.

8.3.3.2 Situation Tracking Prototype

A prototype was developed to track a situation as new input is received. Figure 16 shows the GUI, which requires the initial focal situation and the incoming event. In the figure, the initial focal situation is http://vistology.com/ont/2009/especs/fs0.owl and the incoming event is http://vistology.com/ont/2009/especs/MaintenanceDone.owl . This results in a state transition to ClosedForMaintenance. Although we are interested in tracking the same focal situation as it changes from state to state, a new ontology is generated for the evolved focal situation instead of overwriting the ontology. In this way, we can check the correctness of the morphism by comparing the focal situation in the current state to the focal situation in the former state.

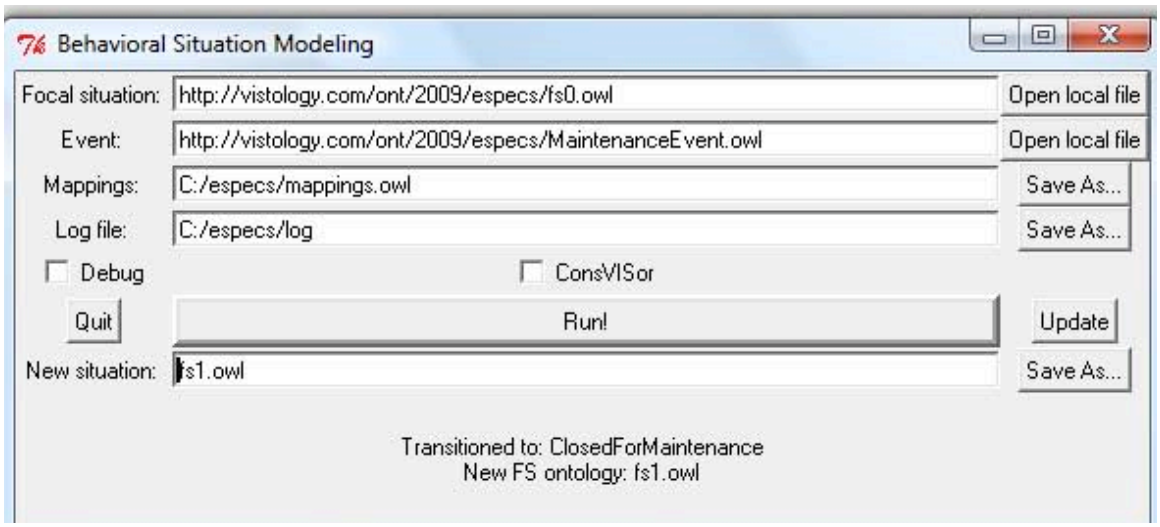


Figure 29 Situation tracking prototype

The GUI and the backend are developed in Python. The backend generates BaseVISor-processable rules from the rule string of the OWL ontologies. Then the BaseVISor inference engine is used to reason with the ontologies. BaseVISor incorporates the axioms and consistency rule for R-Entailment, which is essential for making some inferences. An example of BaseVISor's usage is its inference that two Bridge instances are the same when they have the same unique location value. It also determines the state and event type of the focal situation and incoming events, respectively. The result from BaseVISor indicates to either ignore the event, resulting in no transition, or to transition to a new state.

If there should be a transition, the Python script rewrites the focal situation as specified by the rewrite function in the head of the appropriate event category rule, generating the evolved focal situation in a new ontology. The script also checks whether there is a morphism between the new state and the old state by creating another BaseVISor rule based on the morphism information in the head of the transitioning rule. This morphism rule checks for the existence of certain triples in the former focal situation ontology. The script determines these triples by applying the instance mappings of the morphism to the "axioms" of the new state. If the BaseVISor rule fires, the triples exist in the former focal situation ontology and therefore the "theorems" hold true.

ConsVISor can be used to check the consistency of the focal situations (the initial situation and the generated situations) and the incoming events. This is an important step to verifying that the new focal situation remains consistent.

The situation tracking system can continue accepting incoming events and checking for possible transitions for the focal situation. The mappings files stores the information regarding the instances that refer to the same object; more generally, it stores the "owl:sameAs" information related to the focal situation that BaseVISor inferred, i.e. that two instances, properties or classes are considered as the same despite different names.

The prototype can be expanded or integrated with the modeling tools discussed in the following sections.

9 Tool Support for Behavior Modeling and Tracking

The ontology of abstract behaviors developed in this project is a basis for every tool that we envision for the support of the analyst whose goal is modeling and tracking of situations. In this project we have developed concept of such tools and have prototyped some of their functionality. We can view these tools as extensions to our Situation Awareness Assistant (SAWA) system [76].

9.1 SAWA: A Situation Awareness Assistant

The grand vision of the Semantic Web has led to the development of new technologies intended to serve as the groundwork for its eventual realization. Primary among these technologies is the Web ontology language OWL and its more recent extension in the form of the Semantic Web rule language SWRL [77]. It is fair to say that these technologies represent the cutting edge of practical technologies for realizing intelligent agents and advanced applications. As such, these technologies are applicable to problems well beyond those represented by the Semantic Web proper. VIStology, Inc. is committed to the development of formal yet practical reasoning systems applied to problems in the area of situation awareness and information fusion. Our recent endeavors in this area have been grounded in Semantic Web technologies and we have had occasions to stress them to some of their practical limits. In this project we have used ideas from our Situation Awareness Assistant (SAWA) as the base for the conceptualization of tools for behavior modeling and prediction. As support tools, we have two applications, a consistency checker for OWL documents and an inference engine, BaseVISor.

We started developing SAWA as part of a Phase II SBIR research effort funded by AFRL, Rome. The focus of this effort was the investigation of methods for formally reasoning about (and gaining “awareness” of) real-world situations. The SAWA system is designed to monitor streams of events marked up as OWL annotations arising from an evolving situation. Its primary purpose is to automatically detect the emergence of relevant higher-order relations among the situation’s objects. The relevancy of events, objects and relations is determined relative to a user-defined goal and a corpus of predefined domain knowledge represented in the form of OWL ontologies and SWRL rule sets.

While typical relations in the military context include unit aggregation, composition of the force, and such, the number of potentially relevant relation types is practically unlimited. This presents a great challenge to the developers of situation awareness systems since it essentially means that such a system must be able to track any possible relation. In other words, the relation determination algorithms must be generic, rather than handcrafted for each special kind of relation. One way to address this problem is by using generic reasoning tools, like theorem provers. This, in turn, requires that all information must be represented in a formal language. In the ontology-based approach, domain knowledge is captured by an ontology and the incoming stream of instance data is represented as “annotations” in terms of the domain ontology. The user of the system then can set situation monitoring requirements in terms of “goals” (also referred to as “standing relations”), i.e., special events that the system should monitor and warn the

user whenever the relations relevant to the goals are satisfied or not. Such a system also provides a flexible query language in terms of which the user can ask various questions about current and future situations. A prototype of this tool is currently in the state of validation and testing.

9.2 Extending SAWA to Accommodate Situational Behavior Modeling

In this project we have developed a systematic approach to and conceptualized supporting tools for modeling and handling (monitoring and tracking) various complex behaviors. SAWA collects events and updates its state of knowledge about particular situations that are defined by the Goal. The situation knowledge is updated in response to events from Level 1 processing. At this point, SAWA provides an answer to the goal query - whether a specified *Goal* or any of its *Subgoals* are satisfied or not. Towards this aim, SAWA filters events by focusing only on those that are relevant to the goal. At the same time, SAWA discharges others. In order to monitor behaviors, SAWA must view situations as structured objects with complex behaviors. More specifically, the notion of goal must be refined in order to capture the fact that a goal can be monitored and tracked. Goals must thus be represented as state machines, rather than just queries. Monitoring and tracking a behavioral situation must involve running and updating a number of state machines concurrently.

The first capability that had to be added to the SAWA tool is a tool for managing situations. It requires both capabilities of specifying and editing situations by the human user and providing on-line guidance to the user in terms of inserting new states, actions and events that will cause state transitions, rules for recognition that a given situation is in a particular state, or attributes that could be added to a state.

In order to make the tool operational we envision having a library of templates for behavioral situations and goals. The user should be able to select a template, edit its parameters and its structure with the help from the tool. This minimal capability may be extended to include automatic selection and instantiation of a template by the tool in response to events received by SAWA from another system, e.g., from a Level 1 system, or from a Level 2 system (e.g., a text processing and annotation tool). In particular, the tool should be able to fill in various fields in the template using its OWL based inference capability.

A more advanced capability involves the integration of multiple templates into one. This capability is applicable when a single behavioral template is not sufficient to represent a complex behavior. To accomplish this goal, an operator for combining (any) two templates into one needs to be defined and implemented. The theoretical basis for such an operator comes from category theory. More specifically, this operator is a realization of the *colimit* operation of category theory [78]. Colimit is a generalization of the (shared) union operator of set theory. It allows for combining two structured objects (as opposed to just sets of elements) into one structured object in a consistent way.

Inputs to the tool can come from various sources. One source that has already been mentioned is a Level 1 fusion system. Other sources may be text or speech processing

tools, for instance tools that automatically process intelligence reports. The two types of information, sensory and textual, can be integrated through the use of the ontologies that the system will interact with – the domain ontology and the abstract behavior ontology.
When prototyping behavioral situation modeling, most of our ontology development, situation transition experiment setup, monitoring and testing have been done with lots of *manual* editing and pre-processing. We now have a better understanding of the capabilities and limitations of the existing tools.

Arguably the best tool for ontology development, editing and exploration is Protégé. We use it extensively as a general purpose OWL document editor and processor. Even so Protégé is likely to remain as a preferred OWL editor in our tool chest, we believe more specialized and more productive tools should be available for ontologically based behavioral situation modeling and processing.

One way to improve productivity in this area is by means of specialized and user-friendly templates. The next section shows a few examples of such templates in the context of ontology application and learning. To manage efficiently situation development and tracking, highly productive, flexible, and more user-friendly tools are needed. The key functionality that needs to be addressed include:

- ontology development assistants,
- state chart builders,
- situation state browsers,
- situation type library explorers,
- situation state transition templates, etc.

These functions are integrated into our design of a Visual and Integrated Situation - Model Explorer (VIS-ME) system. Its major components are shown in Figure 30.

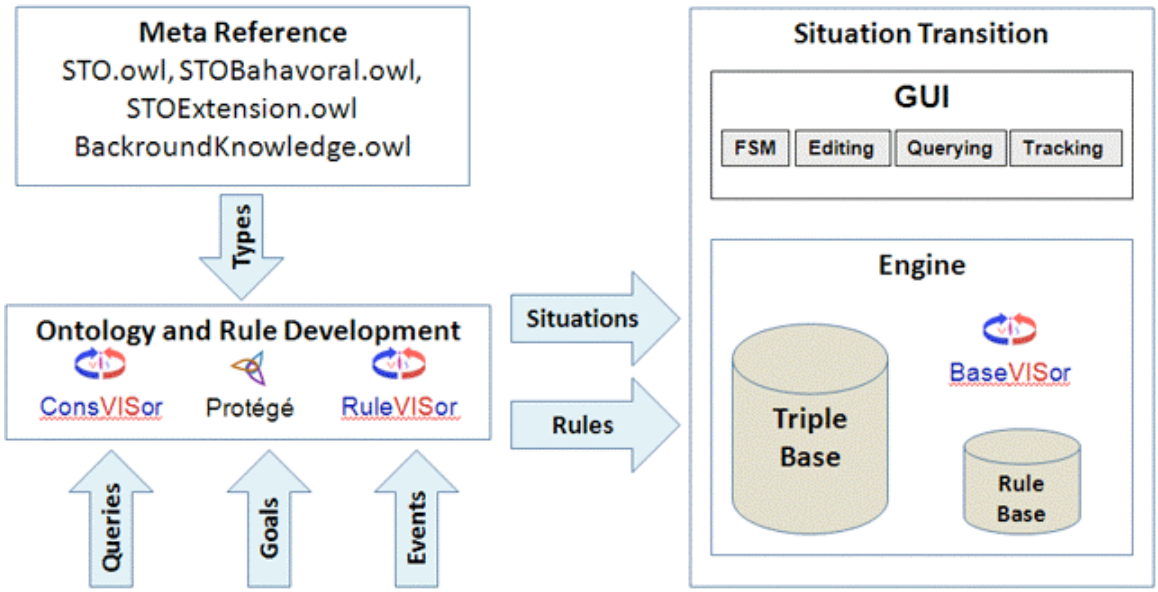


Figure 30 Visual and Integrated Situation - Model Explorer (VIS-ME) system overview.

This system is to provide a rich set of tools for ontology and rule development and pre-processing. It is expected to store all ontologies and rules in a persistent store. Its GUI component should allow for structuring, editing querying, and tracking behavioral situations, including situation state transitions. Last but not least, we assume that such a system should be equipped in a rich library of templates for behavioral situations and goals.

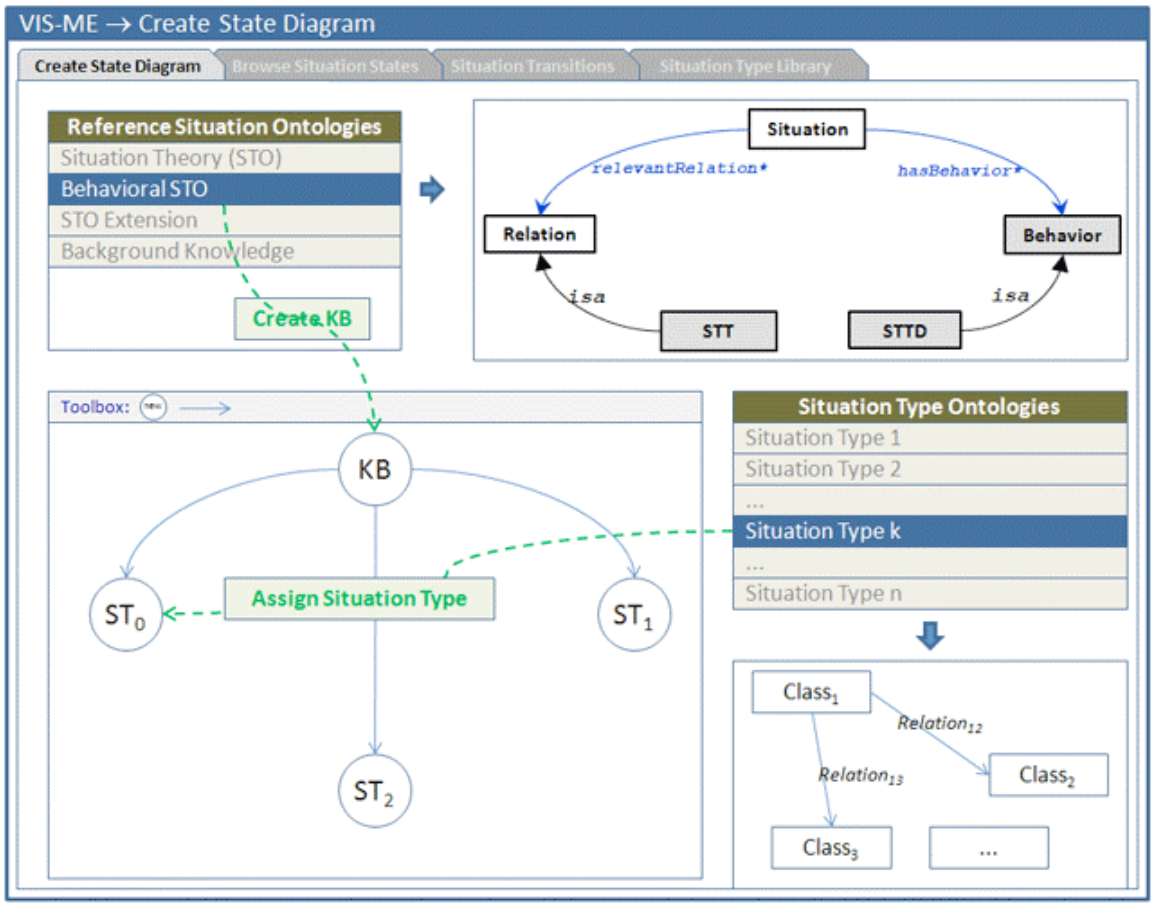


Figure 31 A visual State Diagram Creation tool.

One of the important functions expected of the GUI component is to assist the situation analyst in building state transition diagram (Figure 31). Visual reference and situation type ontology browser should provide an easy-to-follow presentation of situations, objects and relations. The user should be able to drag and drop all entities defining states (ST_k) and contributing to the knowledge base (KB). For clarity, Figure 31 does not show all necessary components. Additional important component would obviously include transition conditions templates based on types of incoming situation events.

Once situation transition states are defined, the system should allow the user for inspecting, verifying and editing the states. Again a visual presentation capability is a key requirement for the system to become ultimately usable and productive. Figure 32 shows

a possible implementation of such a browser. The type component shows OWL classes (types) and relations. Notice that the object relevance is assigned here to classes even so it is referred to as "*relevantIndividual*". In our original Situation Theory Ontology (STO), object and relational relevance could only be defined with respect to individuals (instances of OWL classes). In STO-B, we decided to promote the relevance to the meta level. In particular, regarding situation states, interpreted as situation types, it is necessary to incorporate meta level relevance. Hence, as shown in Figure 32, property "*relevantIndividual*" is defined between OWL classes rather than between OWL individuals.

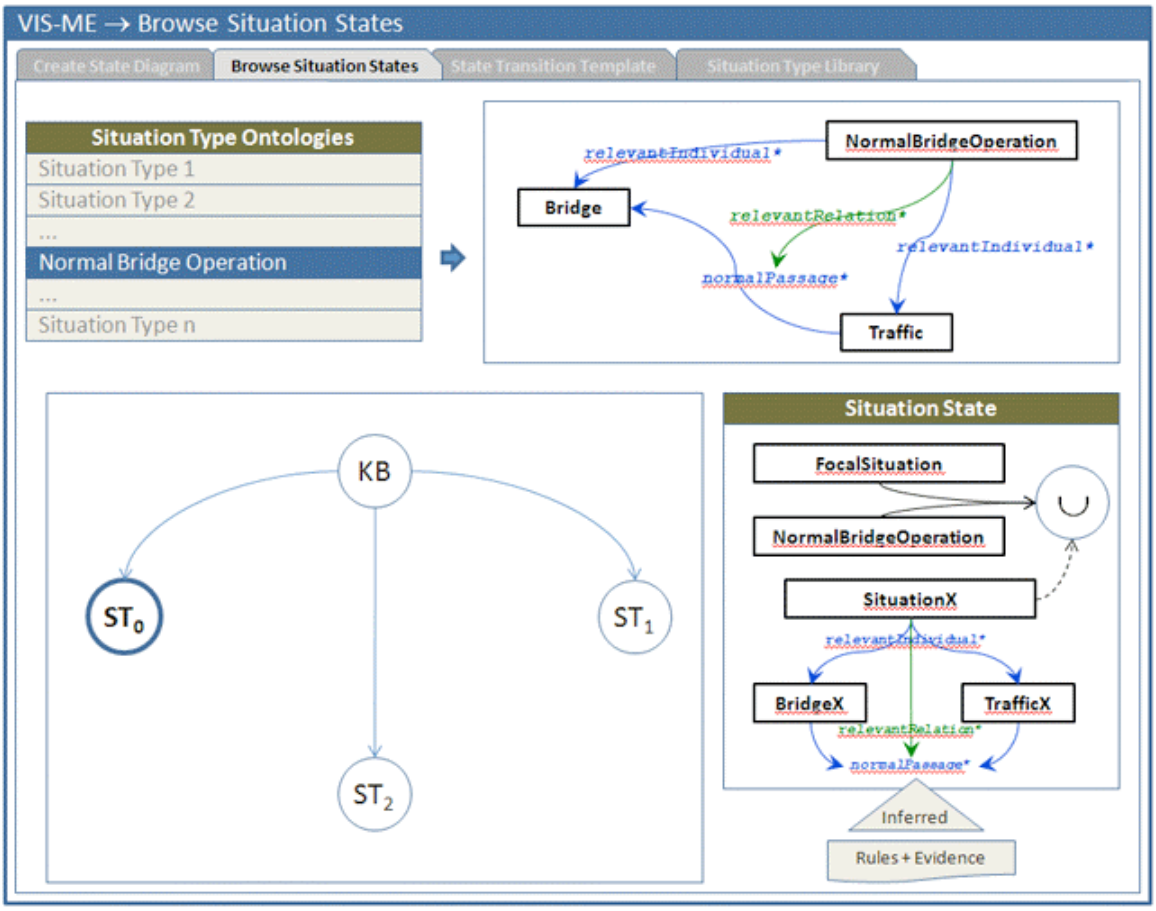


Figure 32 A Situation State Browser subsystem.

An equally important capability of the proposed GUI deals directly with situation transitions described by situation state templates. As shown in Figure 33, such a template shows core elements contributing to a transition from one state to another. The related states along with the event, serving as a transition guard, are shown visually (OWL classes, individuals, and relations). The mappings between entities of the involved situations are represented as a text file. Transition rules are not explicitly shown, although such a functionality is highly desirable and it would certainly be included in the final implementation of the subsystem.

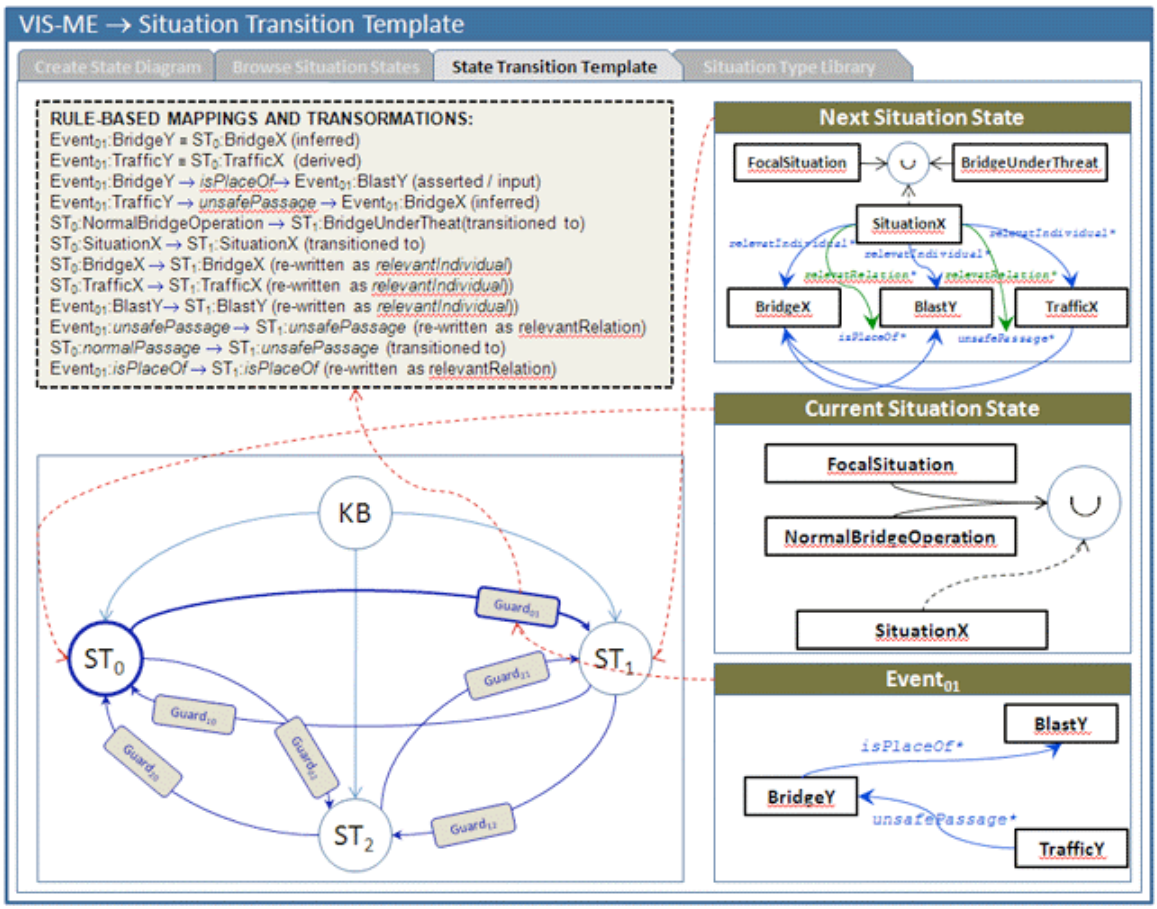


Figure 33 A State Transition Template subsystem.

Every behavioral situation analyst would like to be able to reuse existing and practically verified situation types (states). A draft of such a functionally is shown in Figure 34. The analyst would be able to browse through the situation type ontologies sequentially or by searching situation types associated with a particular object type (OWL class) or relation. Again, it is important to show the result visually (a text/XML view is trivial).

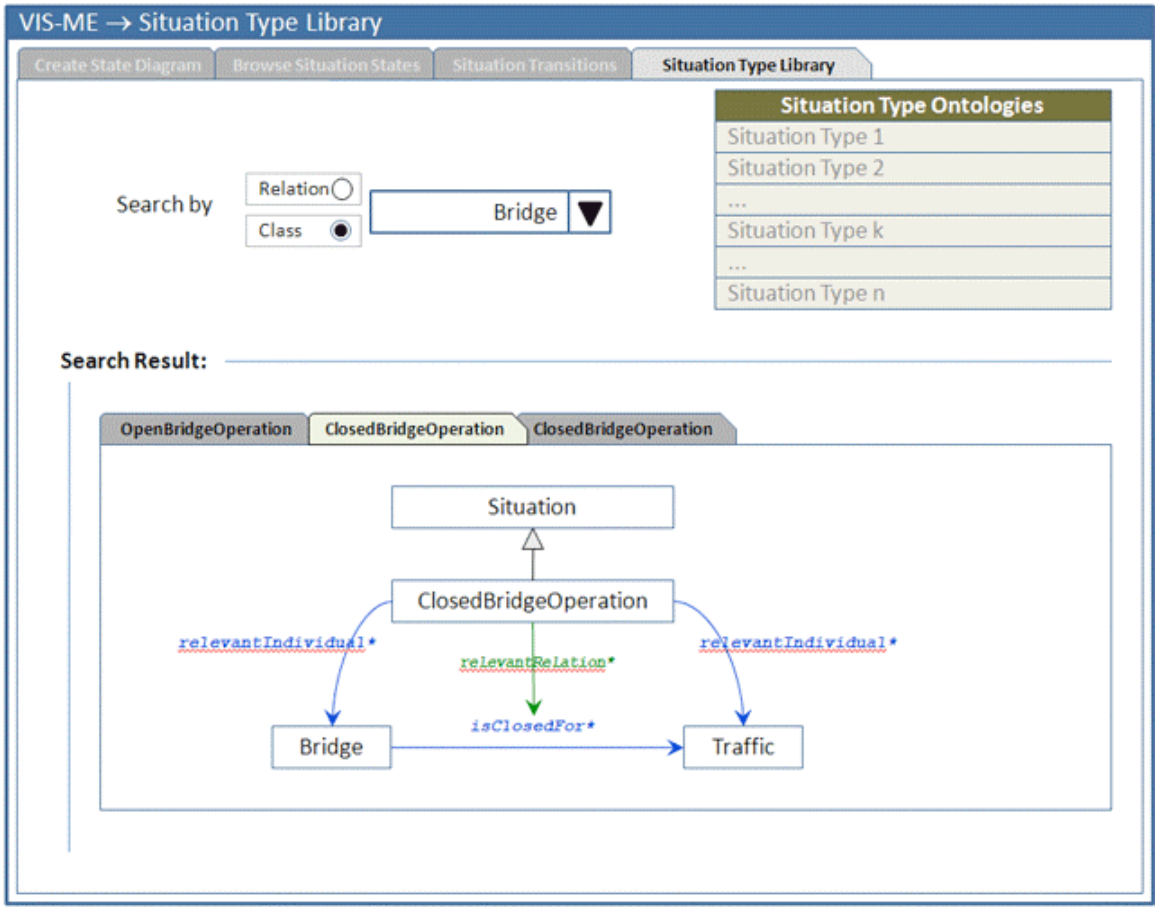


Figure 34 A Situation Type Library subsystem.

A few components of the VIS-ME GUI system presented above by no means exhaust all possible interfaces that situation analyst would desire. Rather our goal was to demonstrate potentially viable patterns and templates enhanced be productive and user friendly graphical tools.

9.3 Learning Behaviors

To present our approach to learning behaviors, we first consider the context for ontology application and learning (see Figure 35). Since in our approach, knowledge about behaviors is to be represented in the same way as ontological knowledge about the domain (i.e., as OWL ontologies/annotations and SWRL rules), the process of learning behaviors from external inputs is a specialization of the process of learning ontologies and rules in general. In this report we discuss the problem of learning ontologies in general (see [79] for a comprehensive review of machine learning approaches used for learning of ontologies). In the behavior learning tool, this approach is refined (and specialized) by incorporating knowledge specific to the Behavior Model discussed in Section 4.

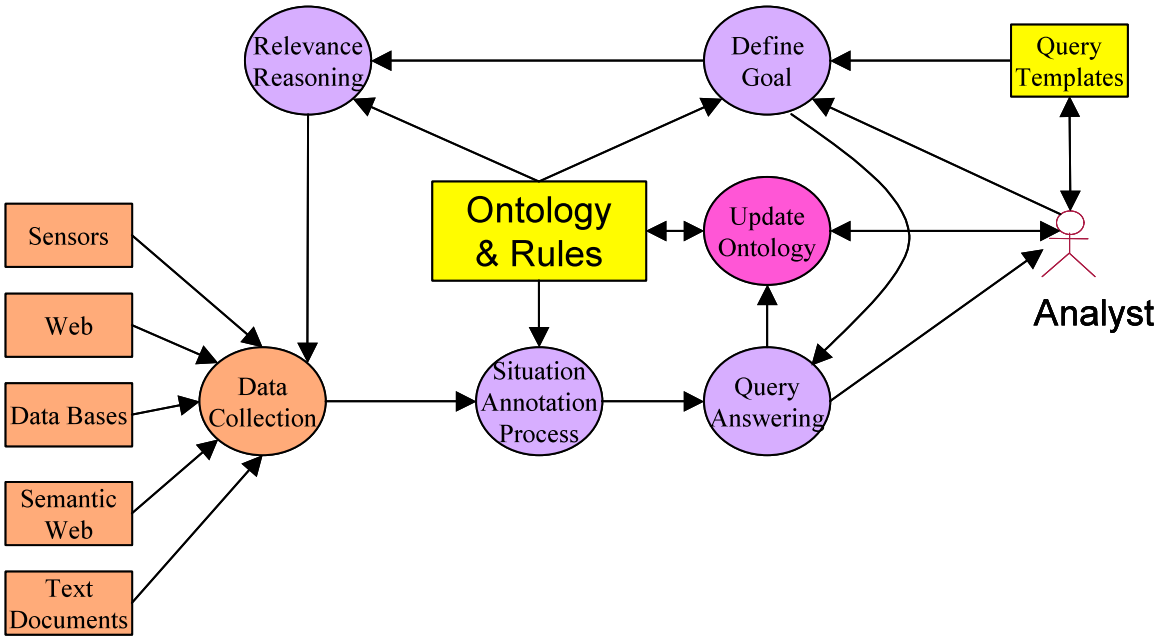


Figure 35 Ontology Based Processing

Figure 35 represents a processing scheme in which an ontology constitutes the central element. We call it **ontology-based processing**. The process involves a human, e.g., an analyst, and some external sources of information, including sensors, the Web (HTML based), the Semantic Web (RDF or OWL based), databases, and text documents. We assume that the pre-processing of the inputs is outside of our system. In other words, some other ontology-based tools perform such operations as the extraction and annotation of information from text. This includes two processes: Data Collection and Situation Annotation in Figure 35. We assume that annotation is also a responsibility of another system or tool. Note that we assume here that the Data Collection process is goal-driven, i.e., we assume that information is brought into the system on demand. In that process, the goal of information retrieval is expressed in terms of an existing ontology and relevance reasoning is performed based upon the goal and the ontology. This is similar to our Situation Awareness Advisor (SAWA). The Query Answering process involves a general purpose, ontology-based reasoner. This process produces an answer to the query formulated by the user (e.g., by an analyst). The Update Ontology process is where ontology editing and learning take place. The details of this process follow.

There many routine ontology development tasks that can be simplified by means of templates. Figure 36 shows an example of a template that can be used to define an Utterance ontology.

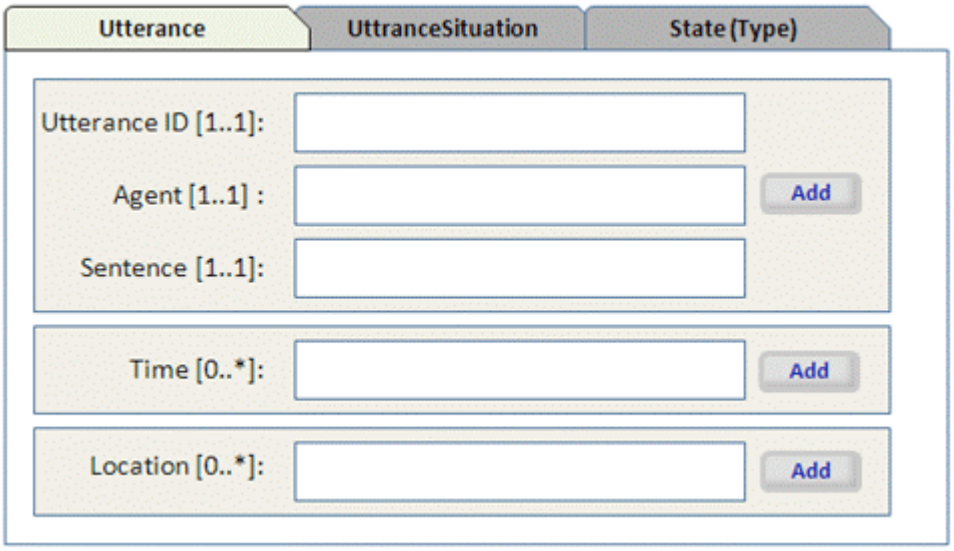


Figure 36 A simple ontology-generating template for an Utterance ontology.

This temple, when filled with UttranceX, AgentX, "Will there be a bridge blown up during the next Boston Marathon?", "2009-04-06"," 2008-06-01" and no Location value will produce the following OWL code:

```
<sto:Utterance rdf:ID="UtteranceX">
 <sto:utteredBy>
  <sto:Agent rdf:ID="InvestigatorX"/>
 </sto:utteredBy>
 <sto:hasAttribute>
  <sto:Sentence rdf:ID="SentenceX">
   <sto:hasAttributeValue>
    <sto:Value rdf:ID="SentenceValueX">
     <sto:attributeValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Will there be a bridge blown up during the next Boston Marathon?
     </sto:attributeValue>
    </sto:Value>
   </sto:hasAttributeValue>
   <sto:hasAttributeValue>
    <sto:Value rdf:ID="NextMarathonTimeValue">
     <sto:attributeValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       2009-04-06
     </sto:attributeValue>
    </sto:Value>
   </sto:hasAttributeValue>
  </sto:Sentence>
 </sto:hasAttribute>
 <sto:hasAttribute>
  <sto:Time rdf:ID="UtteranceTimeX">
   <sto:hasAttributeValue>
    <sto:Value rdf:ID="UtteranceTimeValueX">
     <sto:attributeValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      2008-06-01
     </sto:attributeValue>
    </sto:Value>
   </sto:hasAttributeValue>
  </sto:Time>
 </sto:hasAttribute>
</sto:Utterance>
```

Other our prototypes include templates for the Utterance Situation ontology and for typical states (ontology types) as shown in Figure 37 and Figure 38.

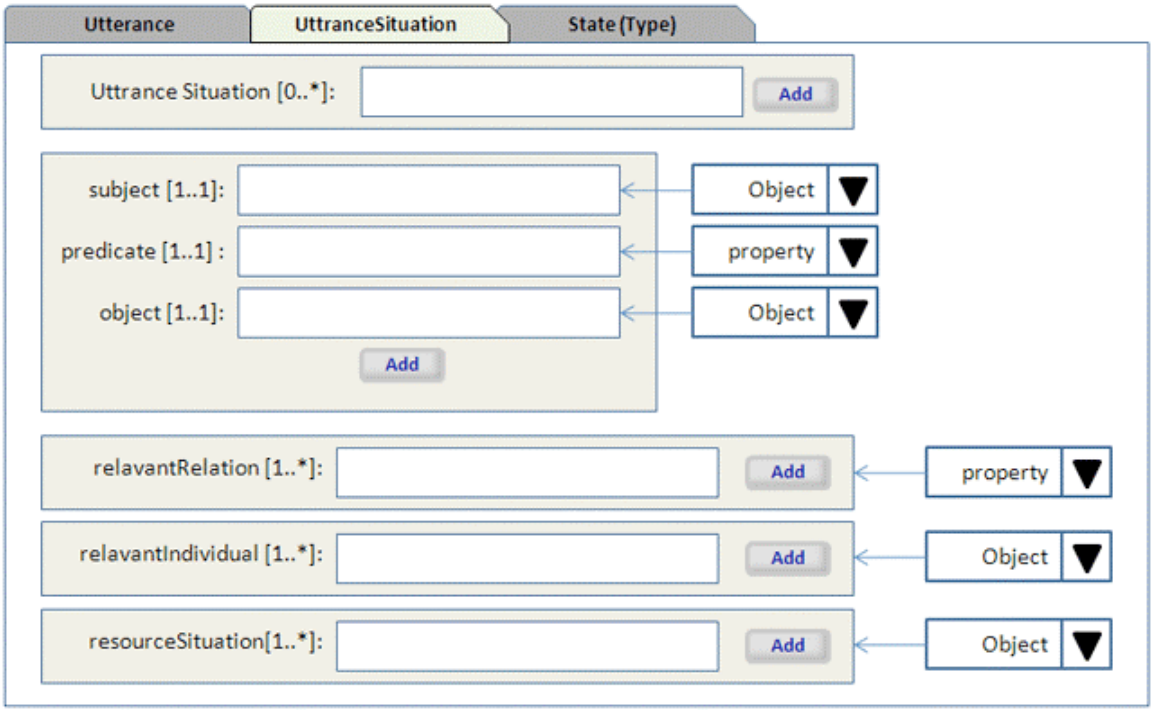


Figure 37 An Utterance Situation generating template.

An example of a document generated with the Utterance Situation generating template is shown at http://vistology.com/ont/2008/STO/BridgeExplosionUtteranceSituation.owl.

A situation type (state) generating template was used to generate three situation type ontologies:
http://vistology.com/ont/2009/espex/BridgeOpen.owl,
http://vistology.com/ont/2009/espex/BridgeClosed.owl,
http://vistology.com/ont/2009/espex/BridgeUnderThreat.owl.

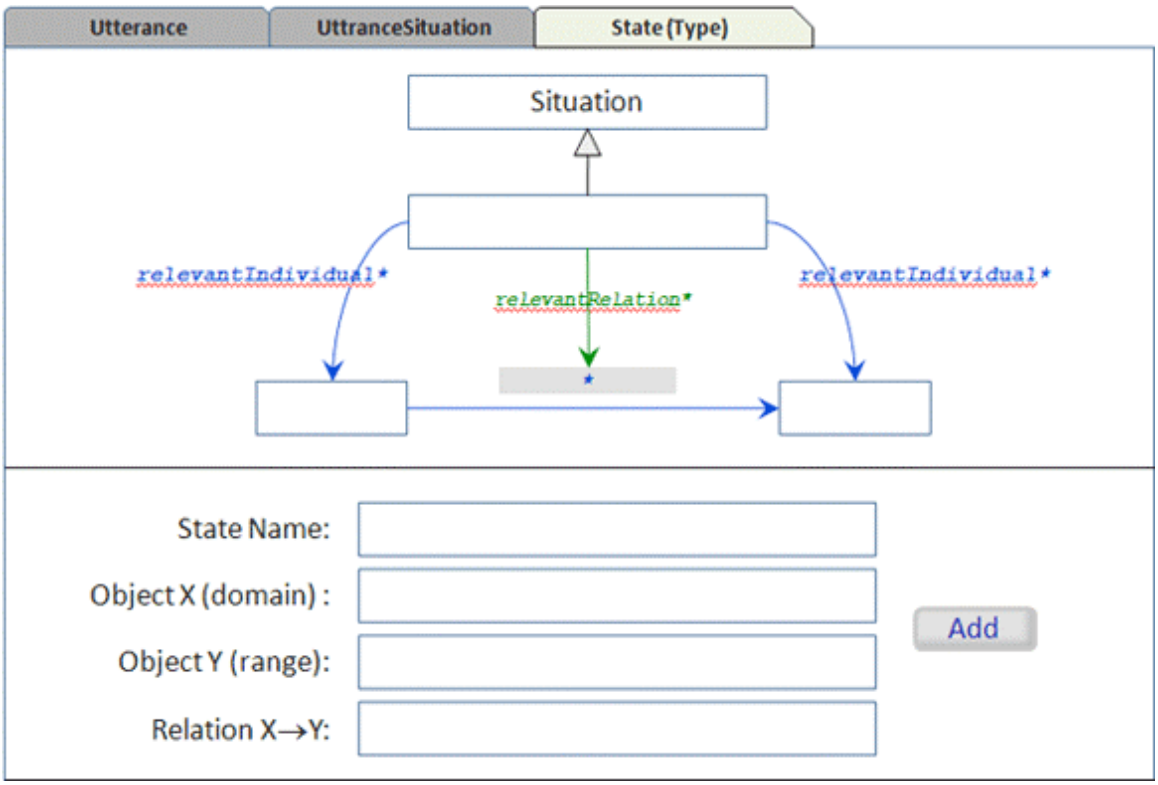


Figure 38 A simple situation type generating template.

Our objective in our future work is to develop a feasibility prototype that implements a learning opportunity identified by the application of our formal framework to SAWA and its extension that covers behavior modeling. Below we discuss different approaches to learning and describe a learning process that could serve as the basis for this prototype. This discussion comes with the caveat that the actual implementation may be very different from this process, depending upon results of earlier tasks.

Machine learning includes *learning from examples*, *reinforcement learning* and *discovery*. In learning from examples a teacher provides examples that are labeled with a concept name and either a *positive* or *negative* label. The learner then adapts the description of a given concept to incorporate positive examples and/or exclude negative examples. In reinforcement learning, the feedback is not classified as positive or negative, but it is measured quantitatively. The learner needs to figure out how to interpret the measure, i.e., whether it should modify the concept description, in which direction and by how much. In discovery, the feedback comes from a monitor that computes some quality measure. Typically this is a measure of similarity of objects that fall in the same concept or dissimilarity between concepts from different classes.

For ontology learning the discovery process seems to be most appropriate. One of the reasons for this is that an ontology is a general-purpose structure that can be utilized in many different ways. Therefore, the coupling of an ontology with any specific task or annotation cannot be too close. Making this coupling too strong would make the learning process too sensitive to single inputs: what is good for one input can prove not so good

for another. Consequently, we will analyze various learning rules in order to find the ones that guarantee both good convergence of the learning process as well as the stability of this process.

The learning process envisioned at this time consists of the following major steps:

1. Receive input from the Query Answering process (this includes the query, also referred to as Goal). Refer to Figure 35.
2. Compute the value of the Evaluation Function (critics) for the current version of the ontology. Refer to Section 9.3.1.
3. Focus on a particular construct in the ontology and apply a transformation operator to the construct. (A number of learning operators will be defined, similarly to inductive learning operators known in machine learning.)
4. Compute the value of the Evaluation Function for the transformed version of the ontology.
5. Perform a search for transformations that improve the ontology in terms of the Evaluation Function (i.e., invoke the above step).
6. Apply the learning rules to select the good transformations.
7. If "good" transformations are found, ask the user to accept or reject the proposed transformation(s).
8. If acceptance is received from the user, update the ontology accordingly.

This is just an outline of the learning cycle. In the project we will develop all the necessary details of this process. Additionally, we need to develop rules (conditions) for the invocation of the learning process. More specifically, not every input should trigger the learning process; this would require too much user involvement. The learning process must be invoked selectively, only when significant improvement in the performance of the ontology-based Fusion 2+ system is expected.

9.3.1 Evaluation Function

While there are many different types of evaluation functions known in the literature on machine learning, a good example that seems like it may have applicability to our work is the utility metric used by Fisher's COBWEB learning system [80]. The basis for the utility metric is the ability to predict properties and property values of new instances in the class. It was originally proposed to capture the essence of human classification [81] (Gluck and Corter, 1985). In OWL ontologies, it would seem that individuals from the same class should be similar in the sense that they should share the same features and the same feature values (intra class similarity). Individuals from different classes should be dissimilar, i.e., they should not have common properties and property values. The same ideas apply to properties and sub-properties as well, i.e., tuples from the same property should have similar characteristics, like domain, range and cardinality constraints. Tuples from different properties, on the other hand, should have different characteristics. The intra class similarity is captured by the probability $P(A_i = V_{ij} \mid C_k)$ being high for a specific value V_{ij} of a property A_i. Similarly, the inter-class dissimilarity is captured by the value of the probability $P(C_k \mid A_i = V_{ij})$ being high. These two measures, when

multiplied and summed over all the classes, give a measure of utility of a given classification: $\sum_{k=1}^{n}\sum_{i}\sum_{j} P(A_i = V_{ij})P(C_k \mid A_i = V_{ij})P(A_i = V_{ij} \mid C_k)$. A metric of this sort could be used by a critic in determining whether to suggest the application of specific generalization or specialization operators.

9.4 Uncertainty Modeling

It is obvious that any behavior modeling and tracking tool needs to be able to accept inputs that carry a degree of uncertainty and be capable of using this kind of information in its processing. The selection of an approach to uncertainty modeling and handling must take into consideration that in our case both quantitative and symbolic information will be processed at the same time. There will be some uncertainty associated with the inputs, but also with the processing rules – i.e., the processing logic. Moreover, uncertainty processing is an extremely computationally intensive task and thus the issue of efficiency of computation must be taken into account when designing such a system.

For representing uncertainty that satisfies the above requirements a number of approaches can be used. One of them would be to convert all the information about the uncertainty into logical statements and then carry out reasoning about the uncertainty within a logical system. A comprehensive theory for this kind of an approach is presented in [82]. While this theory gives very clear definitions and the process has formally stated semantics, the approach does not seem to be applicable due to the high complexity (undecidable) of the logical reasoning problem.

Another typical approach to handling uncertainty is to use Bayesian networks [8]. This approach has been used by many researchers in the information fusion community. There are various problems with this approach. First, the computational complexity of querying a Bayesian network is very high. Second, in order to apply Bayesian reasoning, one first needs to set up the structure of the network and also come up with initial values of the conditional probabilities – but where do these two come from?

We are currently investigating an approach in which the Bayesian network concepts will be combined with some concepts of semantic networks and will be applied in the context of an ontology and a rule base. The paradigm of semantic net processing involves "inference through spreading activation". A piece of evidence, once input into a semantic network, will cause propagation of effects on other nodes that are connected to the node where the information enters the net. Effects are then registered at every network node that has been affected by the spreading, where they are cached for future use. This way of processing partially alleviates the problem of complexity of the computation.

The second question of where the structure of the network comes from is also partially addressed by this approach. Since we are dealing with a rule set and an ontology (which can be viewed as a rule set, too) we can derive the initial structure of the net from the rules and the ontology. We are working on the algorithms for this derivation. The question of where the initial probabilities come from is still open. However, since we

plan to use machine learning, the initial probabilities can be modified in response to feedback that comes from either a human user (user-driven) or from data (data-driven). Currently we are working on this kind of algorithms and plan to use them in this project.

9.5 Behavior Tracking and Prediction

Once instantiated, a model will be updated in response to the incoming events resulting in an update to its *current state*. State transitions will occur in response to incoming events from both Level 1 systems and from Level 2 processing, e.g., intelligence reports that are processed by ontology-based annotation tools. Particular states of the model will be similar in nature to the OWL-based queries. They will be monitored in a manner similar to monitoring situational relations. A query answering mechanism will use the Bayesian reasoning mechanisms described in Section 9.4 to derive most likely transitions. We will investigate various tracking mechanisms. One of the possibilities is to run multiple STDs, a concept similar to multiple-hypothesis tracking known in Level 1 fusion. Trade-offs between accuracy and computation efficiency will be analyzed.

10 Conclusions and Future Directions

In this project we have performed the first formal investigation of developing concepts necessary to define and model behaviors of complex objects in complex situations. The resulting theory of situational behaviors is a significant contribution to the community-owned knowledge of information fusion. In particular, we have provided a clear formal definition and meaning for the term of *situation tracking*. This term has been used in various presentations of information fusion researchers, but we have not seen any formal definition of this term.

Another contribution of this project was the idea of treating situations as objects with associated behaviors. The research in behavioral modeling so far has considered behaviors as either associated with a specific type of object (typically a human or animal, but also a software system) or investigated behaviors of humans within an organization. In our approach, we consider behaviors of situations in which neither the organization nor the type of object are fixed. This kind of approach is necessary for modeling behaviors in the military domain since in this domain all kinds of objects are participants in various behaviors, arranged in all kinds of organizations, and moreover, organizations are objects themselves that also participate in behaviors. The solution of raising the level of abstraction to situations is a novel idea that seems to be appropriate for this complex domain.

The third contribution of this project is the development of a new kind of behavioral modeling paradigm – the ontological approach to behavioral modeling. By using ontological concepts - such as class, sub-class, property and sub-property – we are able to re-use elements of a behavioral model of one situation to be used in modeling another situation that is either more or less general, or related to another situation for which a model has already been developed. This capability adds to both the efficiency and the economy of the tools for behavioral modeling.

Finally, we have investigated human-assisted and data-driven learning applied to behavioral ontologies constructed using OWL and a rule language. We have delineated the space of constructors, operators, critics, and rules for guided learning. Our framework can be applied to the construction of learning capabilities of the future situation modeling and tracking tools.

The formal framework for behavior modeling provides us, and others, with the means for investigating the various ways of modeling and learning behaviors. It provides proofs for the concepts introduced in Section 7. A progression of tools has been conceptualized and analyzed within a number of behavioral scenarios. The results of this project constitute a clear specification of prototypes that will be developed in the continuation work after this project. Behavior learning will be a major focus of the future projects. This project has provided a language for analysis and outlined a basic analytic process for learning behavioral ontologies. From this, new learning opportunities and techniques will be elucidated thereby permitting the development of improved Fusion 2+ systems.

10.1 Addressing the Issues of Scalability, Maintainability, Robustness and General Applicability of the Ontological Approach

In this section we address a number of issues that can be viewed as potential problems with the ontological approach to behavioral modeling developed in this project. These include 1) Knowledge Elicitation and Representation, 2) Scalability to Real-world Problems, 3) Robustness and 4) General Applicability. Some of the means for addressing these problems have been investigated in this project, while the others have been addressed in our previous projects or are being addressed in our current projects.

10.1.1 Knowledge Elicitation and Representation

Since the approach presented in this report is heavily dependent on the availability of domain knowledge represented in computer-processable form, the process of knowledge capture and representation may be perceived as a factor that needs to be resolved for this approach to be practical. In particular, the user community would need to have tools that would allow them to input the knowledge directly, without requiring a highly specialized (in mathematics, logic, AI and computer science) knowledge engineer to work with an SME to achieve such a goal. This problem is especially important when the knowledge base grows larger and thus making it difficult to add new knowledge that does not overlap or conflict with existing knowledge. This problem can also be viewed as the scalability of the knowledge representation. To address these problems we foresee the means listed below.

1. **Ontologies**: An ontology provides a dictionary of terms that the analyst can choose from in order to construct a situation-specific ontology.
2. **Ontology libraries**: An ontology from a collection of ontologies developed for various aspects of the domain can be chosen by the analyst to serve as a base in developing a new ontology for a specific scenario.

3. **Ontology development support tools:** Tools, like the consistency checking ConsVISor tool developed by VIStology, will give feedback to the analyst on the consistency of the ontology being developed. The use of such a tool will streamline the ontology development process.
4. **Rules development support tools**: Tools, like the RuleVISor tool developed by VIStology, will support the analyst in developing domain knowledge, including drag-and-drop operations and consistency checking. The use of such tools will streamline the process of rule development. Further improvement will be achieved by extending or even transforming RuleVISor to allow the user to provide knowledge in a high-level situation awareness specific language.
5. **Ontology integration support tools**: Tools that support ontology integration (fusion) have been investigated in this project. Such tools will help to map terms from one ontology to the other and then to create a new ontology using the category-theoretical colimit operation.
6. **Ontology extension support tools**: A tool that includes the operators that can be used for extending ontologies to accommodate new situations will be investigated in our future projects. In this case a base ontology exists; the analyst provides semi-ontological annotations in which some of the terms are from an existing ontology while some other terms are created by the analyst on the fly. The tool would help to map the terms and to extend the ontology in order to accommodate the new terms.
7. **Ontology learning support tools**: A tool that supports ontology learning has been investigated in this project. In a use case for this tool it is assumed that an ontology exists; the analyst provides examples of semi-ontological descriptions. The learning algorithm then infers ontology extensions.

10.1.2 Performance Scalability

One of the problems with logic-based systems is that the inference engines don't perform well when confronted with real-world levels of data. Both the issue of memory requirements and CPU processing requirements need to be addressed. Some of the ways of addressing these problems are listed below.

1. **Power computers**: More powerful computers (with multiple CPUs or computer clusters) with large amounts of memory can be used. This is the approach currently used by financial systems that must process gigabytes of information in real-time and it may be useful for ontology based processing.
2. **Specialized databases**: Databases specialized for ontologies and rule bases can provide the large storage capability while keeping access time low. There are these kinds of databases on the market already and more will be developed as the development of the semantic web concept progresses.
3. **Specialized inference engines**: Inference engines specialized for ontology based processing can provide significantly improved inference efficiency. BaseVISor, the inference tool being developed at VIStology is an example of such an engine.

10.1.3 Robustness

Rule based systems suffer from the problem of being brittle, i.e., all the preconditions of an application of a rule must be met, or otherwise the rule will not fire and thus the system will not provide any result to the user. Some of the ways this problem has been addressed by the ontology based approach and by incorporating uncertainty into knowledge based processing are listed below.

1. **Formal Representation**: The formal representation of ontological knowledge allows inference engines to draw conclusions not only from domain specific rules, but also by using the power of the ontology representation language, like OWL, in which such concepts like sub-classing, inheritance, transitivity and functionality of relations, cardinalities of properties can be used in the inference process. For instance, if a conclusion cannot be drawn based upon knowledge for a specific concept, there still may be some knowledge associated with a super-class that can be used in the inference. This kind of capability requires a full OWL specific reasoning engine. While our BaseVISor at this time is not a full OWL reasoner, we plan to extend in this direction.
2. **Semantic Web**: The main idea of the Semantic Web activity is to develop systems that are interoperable, i.e., able to share knowledge. In case a specific system does not have sufficient knowledge to solve a given problem, it can actively search the semantic web for additional information. In this sense this capability addresses the issue of robustness of an ontology based system. While the active search for knowledge has not been incorporated into the VIStology developed systems, we are investigating such an approach in one of our current projects and plan to have such a capability at a later time.
3. **Incorporation of Uncertainty**: The incorporation of uncertainty into the knowledge representation and inference mechanisms will provide improved robustness by permitting the system to work with data that may be less than certain, i.e., it will allow to derive conclusions even when there is not sufficient evidence to support such a conclusion with a high degree of certainty.
4. **Abductive Reasoning**: The ability to recognize partially matched rules means the system can still respond to a situation even if it does not perfectly match what was expected. We plan to extend BaseVISor to enable it to detect partially matched rules.
5. **Adaptive Systems**: The incorporation of mechanisms for monitoring and controlling the inference process will help ensure that an “any time solution” is found within the prespecified time constraints, even if the solution is not optimal. We have partially addressed this issue in our SAWA system by implementing some inference monitoring mechanisms. We plan to eventually extend this capability to also include adaptive control of the inference engine based upon the feedback received from the monitoring process.

10.1.4 General Applicability

A typical computer program includes both some quantitative processing and some logic. Current ontology based systems tend to use mainly, if not exclusively, logical inference.

Purely logic based processing may not be appropriate, or sufficient, for various problems and domains. The following means can be viewed as ways to address this problem.

1. **Quantitative Data Types**: The semantic web representation languages (OWL and SWRL) provide means for representing quantitative data types as well as provide operations for the processing of such data. All VIS tools, like ConsVISor and BaseVISor, incorporate these data types though a subsystem called KRDF.
2. **Procedural Attachments**: In cases where purely logic based (rules and ontology based) solutions are not applicable, a mechanism for extending the capabilities of the system with "procedural attachment", i.e., the invocation of numerical procedures from within the logical inference engine, will be provided. Support for procedural attachments is built into BaseVISor.

11 References

1 F. E. White, Jr. A model for data fusion. Proceedings of the first National Symposium on Data Fusion, vol. 2, 1988.

2 A. N. Steinberg and C. L. Bowman. Revisions to the JDL Data Fusion Model. In D. L. Hall and J. Llinas (Eds.), Handbook of Multisensor Data Fusion, CRC Press, pages 2.1-2.19, 2001.

3 M. Endsley and D. Garland, Situation Awareness, Analysis and Measurement, Lawrence Erlbaum Associates, Publishers, Mahway, New Jersey, 2000.

4 WordNet: a lexical database for the English language. Princeton University, January 2005.

5 Webster's New World Dictionary of the American Language. Second College Edition. Simon and Shuster, 1982.

6 UML 2.0 Superstructure Specification. OMG. http://www.omg.org/docs/ptc/03-08-02.pdf, Document -- ptc/03-08-02, January 2005.

7 R. A. Howard and J. E. Matheson. Influence diagrams. In Readings on the Principles and Applications of Decision Analysis, pages 721–762. Strategic Decisions Group, 2003.

8 J. Pearl. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, 1988.

9 V. Akman and M. Surav. Steps toward formalizing context. AI Magazine, 17(3):55–72, 1996.

10 J. McCarthy. Generality in artificial intelligence. Communications of the ACM, 30(12):1030–1035, 1987.

11 F. Sowa, J. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks/Cole, 2000.

12 F. Sowa, J. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, 1984.

13 J. Barwise. Scenes and other situations. Journal of Philosophy, 78(7):369–397, 1981.

14 J. Barwise and J. Perry. Situations and Attitudes. MIT Press, Cambridge, MA, 1983.

15 J. Barwise. The Situation in Logic. CSLI Lecture Notes 17, 1988.

16 K. Devlin. Logic and Information. Cambridge University Press, Cambridge, U.K., 1991.

17 W3C. Semantic Web Activity, 2006. http://www.w3.org/2001/sw/.

18 RDF. Resource description framework (RDF) model and syntax specification, Feburary 1999. http://www.w3.org/TR/REC-rdf-syntax.

19 W3C. Web Ontology Language Reference OWL, 2004. http://www.w3.org/2004/OWL/.

20 T. McClanahan, J. Feinberg, P. Goalwin and P. Blemberg. Human and Organizational Behavior Modeling (HOBM): Technology Assessment. MSIAC Project MS-00-0019/0028, Modeling and Simulation Information Analysis Center, 2001.

21 G. Mocko, R. Malak, C. Paredis and R. Peak. A knowledge repository for behavior models in engineering design. Proceedings of DETC '04: 24th Computers and

Information Science in Engineering Conference September 28 – October 3, 2004, Salt Lake City, Utah.

22 OWL Web Ontology Language Overview. http://www.w3.org/TR/2003/PR-owl-features-20031215/.

23 M. M. Kokar, C. J. Matheus, and K. Baclawski. “Ontology-based situation awareness.” Information Fusion, 10:83-98, 2009.

24 C. Matheus, M. Kokar and K. Baclawski, A Core Ontology for Situation Awareness. In Proceedings of FUSION’03, Cairns, Queensland, Australia, July 2003.

25 Data fusion lexicon. Technical report, The Data Fusion Subpanel of the Joint Directors of Laboratories, Technical Panel for C3, 1991.

26 Merriam-Webster Online, 2004. Available at: http://www.m-w.com/.

27 L. vonMises. Human Action: A Treatise on Economics. Fox & Wilkes, 1997.

28 M. Bunge. Treatise on basic philosophy. III: Ontology: The furniture of the world. Reidel, Dodrecht, 1977.

29 In F. Baader, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, The Description Logic Handbook. Cambridge University Press, 2003.

30 I. Horrocks and F. Patel-Schneider, P. A proposal for an OWL Rules Language. In Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), pages 723–731. ACM, 2004.

31 C. J. Matheus, K. Baclawski, and M. M. Kokar. BaseVISor: A Triples-Based Inference Engine Outfitted to Process RuleML & R-Entailment Rules. In RuleML-2006, Rules and Markup Languages for the Semantic Web, Second International Conference, 2006.

32 H. ter Horst. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Proc. of the Fourth Int’l Semantic Web Conference. Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 668–684, 2005.

33 W3C. SPARQL Query Language for RDF. W3C Candidate Recommendation, 2006. http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/.

34 B. Jacobs, Introduction to Coalgebra. Towards Mathematics of States and Observations, book draft at http://www.cs.ru.nl/ bart.

35 D. Harel, Stetecharts: a Visual Formalism for Complex Systems, Science of Computer Programming, 8(198) 231-274.

36 J. Barwise, J. Seligman, Information Flow, Cambridge University Press, 1997.

37 Y. Kalfoglou, M. Schorlemmer, IF-Map: An Ontology-Mapping Method based on Information-Flow Theory, preprint.

38 E. H. Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses. In Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC), Granada, Spain, May 1998.

39 D. L. McGuinness, R. Fikes, J. Rice and S. Wilder. The Chimaera Ontology Environment. In Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000). Austin, Texas July 2000.

40 N. F. Noy and M. A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In 17th National Conference on Artificial Intelligence (AAAI'00), Austin, Texas, July 2000.

41 Protégé Ontology Editor and Knowledge Acquistion System. protege.stanford.edu/

43 H. Chalupsky. OntoMorph: A translation system for symbolic knowledge. In KR2000: Principles of Knowledge Representation and Reasoning, A. G. Cohn, F. Giunchiglia, and B. Selman (Eds.) Morgan Kaufmann, pp. 471–482, 2000.
44 G. Stumme and A. Maedche. FCA-Merge: Bottom-up merging of ontologies. In Proceedings of the 17th International Joint Conference on Artificial Intelligence, (IJCAI ’01), Seattle, WA, Aug. 2001.
45 B. Ganter and R. Wille. Formal Concept Analysis: Mathematical Foundations. Springer: Berlin, Germany, 1999.
46 Ontology Alignment Evaluation Initiative. http://oaei.ontologymatching.org/
47 P. Wang, B. Xu. Lily: Ontology Alignment Results for OAEI 2008. In Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, Oct. 2008.
48 M. M. Kokar, J. A. Tomasik, and J. Weyman. Data vs. decision fusion in the category theory framework. In Proceedings of FUSION 2001 - 4th International Conference on Information Fusion, Vol .1, pages TuA3-15 - TuA3-20, 2001.
49 M. M. Kokar and Z. Korona. A formal approach to the design of feature-based multi-sensor recognition systems. International Journal of Information Fusion, 2 (2):77-89, 2001.
50 C. J. Matheus, K. Baclawski and M. M. Kokar, Derivation of ontological relations using formal methods in a situation awareness scenario, In Proc of SPIE Conference on Mulitsensor, Multisource Information Fusion, pages 298-309, April 2003.
51 K. Baclawski, M. Kokar, J. Letkowski, C. Matheus and M. Malczewski, Formalization of Situation Awareness, Eleventh OOPSLA Workshop on Behavioral Semantics, pp. 1-15, November, 2002.
52 C. Matheus, K. Baclawski, M. Kokar, and J. Letkowski, Constructing RuleML-Based Domain Theories on top of OWL Ontologies, Proc. of the RuleML Workshop at the International Semantic Web Conference, Sanibel Island, Florida, October 2003.
53 OMG Standard Specifications. UML 2.0 Superstructure specification. http://www.omg.org/technology/documents/formal/spem.htm.
54 Baclawski, K., Kokar, M. M., Waldinger, R. and Kogut, P. A. Consistency Checking of Semantic Web Ontologies. 1st International Semantic Web Conference (ISWC)}, Lecture Notes in Computer Science, LNCS 2342, Springer, pp. 454--459, 2002.
55 M. Kokar, Final Report: A Formal Approach to Multi-Node Intrusion Detection, AFRL Contract: F30602-99-C-0040, September 1999.
56 M. M. Kokar, J. A. Tomasik, and J. Weyman. Data vs. decision fusion in the category theory framework. In Proceedings of FUSION 2001 - 4th International Conference on Information Fusion, Vol .1, pages TuA3-15 - TuA3-20, 2001.
57 M. M. Kokar and Z. Korona. A formal approach to the design of feature-based multi-sensor recognition systems. International Journal of Information Fusion, 2 (2):77-89, 2001.

58 M. M. Kokar, M. D. Bedworth, and K. B. Frankel. A reference model for data fusion systems. In Sensor Fusion: Architectures, Algorithms, and Applications IV, pages 191-202. SPIE, 2000.
59 J. Smith, M. Kokar, and K. Baclawski. Formal verification of UML diagrams: A first step towards code generation. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, Practical UML-Based Rigorous Development Methods Countering or Integrating the eXtremists, pp 224-240,October 2001.
60 K. Baclawski, M. Kokar, J. Smith, and J. Letkowski, Consistency Checking of RM-ODP Specifications, ICEIS 2001, International Conference on Enterprise Information Systems, Setúbal, Portugal, 7-10 July, 2001.
61 J. Li, M.M. Kokar, and J. Weyman. Incorporating uncertainty into the formal development of the fusion operator. In Proceedings of the Second International Conference on Information Fusion, Vol.1, pages 125-132, 1999.
62 D.A.G. Ercolini and M.M. Kokar. Desktop agent manager (DAM):Decision mechanism. International Journal of Human-Computer Interaction, Vol.9, No.2:133-149, 1997.
63 K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M.Aronson. Extending UML to support ontology engineering for the Semantic Web. In M. Gogolla and C. Kobryn, editors, Fourth International Conference on The Unified Modeling Language, volume 2185, pages 342-360. Springer-Verlag, Berlin,October 2001.
64 M. M. Kokar and J. Wang Using Ontologies for Recognition: An Example. Proceedings of the 5th International Conference on Information Fusion, pages 1324 - 1343, 2002.
65 M. M. Kokar and J. Wang An Example of Using Ontologies and Symbolic Information in Automatic Target Recognition. In Sensor Fusion: Architectures, Algorithms and Applications VI, pp. 40-50, Vol. 4731m Proc. SPIE, 2002.
66 S. A. DeLoach and M. M. Kokar. Category theory approach to fusion of wavelet-based features. In Proceedings of the Second International Conference on Information Fusion,Vol.1, pp 117-124, 1999.
67 M. M. Kokar and M. K. Malczewski. Relations among wavelet coefficients as features for ATR. In Sensor Fusion: Architectures, Algorithms, and Applications V, pages 244-254. SPIE, 2001.
68 Dusko Pavlovic and Douglas R. Smith. Software Development by Refinement. To appear in, UNU/IIST 10th Anniversary Colloquium, Formal Methods at the Crossroads: From Panaea to Foundational Support, Springer-Verlag, 2003.
69 Specware 4.2 Language Manual, 2009. Available at: http://specware.org/documentation/4.2/language-manual/SpecwareLanguageManual.html
70 M. Anlauff, D. Pavlovic, D. Smith, Specification-Carrying Software: Evolving Specifications for Dynamic System Composition, Kestrel Institute preprint, 2004.
71 M. Kokar, J. Letkowski, C. Matheus, Situation Tracking: The Concept and a Scenario, VIStology Inc, Preprint, 2007.
72 M. Kokar, C. Matheus, K. Baclawski, Ontology-based situation awareness. Journal of Information Fusion, Vol. 10, pages 83-98, 2009.

73 M.R. Genesereth, N.J. Nilsson, Logical Foundations of Artificial Intelligence, Morgan-Kaufman, 1987.
74 M. E. Stickel, R. J. Waldinger, M. Lowry, T. Pressburger, and I. Underwood, "Deductive composition of astronomical software from subroutine libraries," Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12), Nancy, France, pp. 341-355, Jun. 1994.
75 OWL/RDF/RDFS Reasoning Capabilities, http://ksl.stanford.edu/software/JTP/doc/owl-reasoning.html
76 C. Matheus, M. Kokar, K. Baclawski, J. Letkowski, C. Call, M. Hinman, J. Salerno and D. Boulware, SAWA: An Assistant for Higher-Level Fusion and Situation Awareness. In Proceedings of SPIE Conference on Multisensor, Multisource Information Fusion, Orlando, FL., March 2005.
77 SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.daml.org/2003/11/swrl/.
78 B. C. Pierce. Basic Category Theory for Computer Scientists. MIT Press, 1991.
79 A. Gómez-Pérez and D. Manzano-Macho "A survey of ontology learning methods and techniques, "http://ontoweb.aifb.uni-karlsruhe.de/Members/ruben/Deliverable%201.5, Universidad Politécnica de Madrid, 2003.
80 Douglas H. Fisher, "Knowledge Acquisition through Incremental Conceptual Clustering", In Readings in Machine Learning, J.W. Shavlik and T.G. Dietterich (Eds.), Morgan Kaufman, 267-283, 1990.
81 Gluck, M. A. and Corter, J. E., "Information, uncertainty, and the utility of categories." Proceedings of the Seventh National Conference of the Cognitive Science Society, Irvine, CA, Lawrence Erlbaum Associates, 283-287, 1985.
82 F. Bacchus. Representing and Reasoning with Probabilistic Knowledge. MIT Press, Cambridge, MA, 1990.